1991 GRAPHICS AND WINDOWING ISSUE

January/February 1991 Volume 9.1

Programmer's Journal

95 Canada $5.95

TM

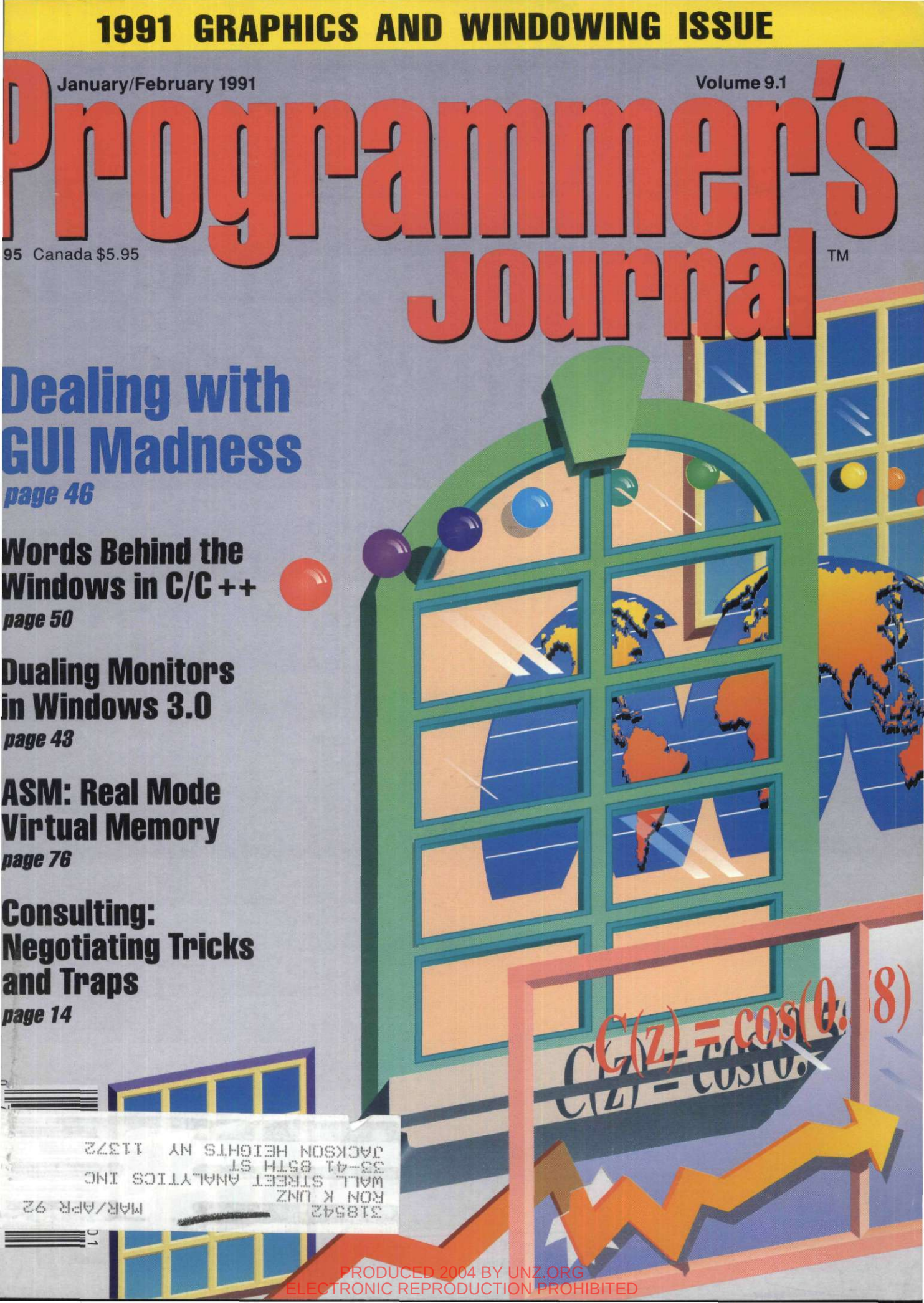
Dealing with GUI Madness
page 46

Words Behind the Windows in C/C++
page 50

Dualing Monitors in Windows 3.0
page 43

ASM: Real Mode Virtual Memory
page 76

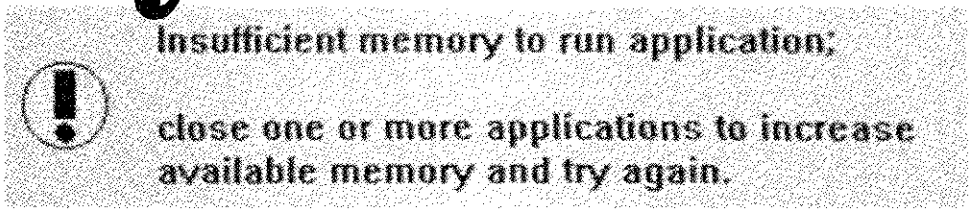
Consulting: Negotiating Tricks and Traps
page 14



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 and 486 Windows users:

DOS Lives. Despite what you may have read.



Windows 3.0 may have been a big step forward for some programs.

But it was a big step backward for DOS. Suddenly, it was 1987 all over again. Not enough room for DOS programs to run because TSRs, utilities, drivers and buffers were taking up room your DOS programs need.

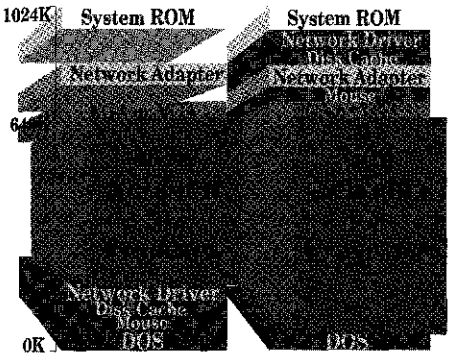
QEMM 5.1 to the rescue

Now, we've updated QEMM to provide additional memory for DOS programs within Windows 3.0.

QEMM 5.1 works with the built-in capabilities of your 80386 or i486 processor to find and recover unused memory segments. As you can see in the chart, there are gaps in your PC's memory usage above 640K. QEMM 5.1 fills those gaps and provides more room for your DOS programs to run.

And of course, QEMM 5.1 still works with DOS when you're not running Windows. You get all the same benefits: up to 130K more memory to run the new generation of memory-hungry programs; space for larger spreadsheets and database files.

When Windows 3.0 says there's not enough room to run your DOS programs, it's just trying to tell you it needs QEMM 5.1.



Before *After*

Unused memory is like hidden gold in your PC. QEMM finds it and makes it available.

Introducing DESQview 2.3 and DESQview 386 2.3 for users of Windows 3.0

They said it couldn't be done, but DESQview 2.3 can run Windows 3.0 programs. Not just in Windows "Real mode" but in "Standard" mode. That means programs can be up to 16MB.

And it can run DOS programs and DOS-extended programs i.e., 1-2-3 Release 3, side-by-side.

DESQview 386 2.3 does all that and more. It lets you run 386 DOS extended programs like AutoCAD 386 and IBM Interleaf side-by-side.

DOS, extended DOS; Windows—whatever standards you set, we will support. We're committed to helping you get the most out of your hardware and software today. And tomorrow.

QEMM-386 System Requirements: 80386-based PCs and PS/2s and PCs with 80386 add-in boards. Operating system: PC DOS 2.0-4.0, MS DOS 2.0-3.3, Windows 3.0. Conventional memory requirement 1.5K.
DESQview System Requirements: IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286, or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage; EMS 4.0 expanded memory boards • Disk: two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM PS/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or compatible • Operating System: PC-DOS 2.0-4.0; MS-DOS 2.0-3.3 • Software: Most PC-DOS and MS-DOS programs; programs specific to Microsoft Windows 1.02-2.1, GEM 1.1-3.0, IBM TopView 1.1 • Media: DESQview is available on either 5-1/4" or 3-1/2" floppy diskette.

Trademarks: Windows, MS-DOS: Microsoft Corporation; PS/2, Interleaf, TopView: IBM Corporation; 80386, i486, AboveBoard: Intel Corporation; 1-2-3: Lotus Development Corporation; AutoCAD 386: Autodesk, Inc; RAMpage: AST Research; Hercules; Mouse Systems; Hayes; GEM: Digital Research, Inc.

©1990 Quarterdeck Office Systems

Yes! I need increased DOS productivity in Windows!

Payment method ☐ Visa ☐ MasterCard

Expiration _____/_____

Card # _____

Name _____

Address _____

City _____

State ____ Zip _____

Qty	Product	5-1/4	3-1/2	Each	Totals
	QEMM 386 5.1			$99.95	
	DESQview 386 2.3 (includes QEMM)			$219.95	
	DESQview 2.3			$129.95	
				Shipping & Handling $5 in USA/$10 outside USA	
				California Residents add 6.75%	
				Grand Total	

Please allow 3 weeks for delivery

Quarterdeck

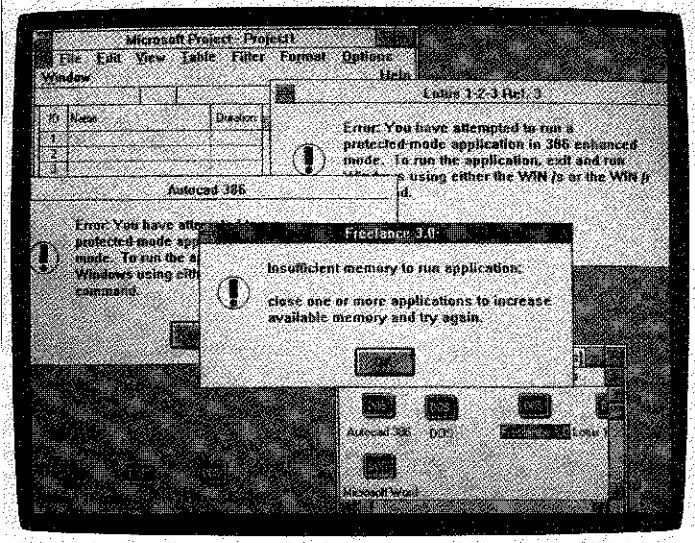
Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Circle **147** on Reader Service Card

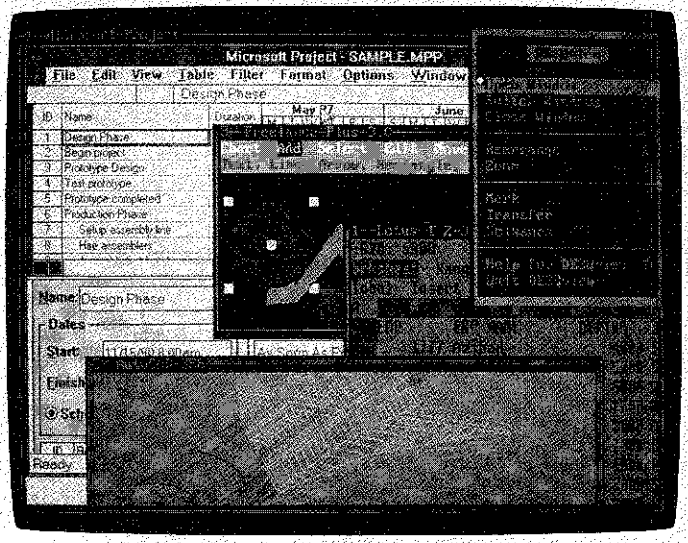
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Is Windows 3.0 the end of DESQview™?

Not if you're still using DOS programs.


Microsoft Project - Project1
File Edit View Table Filter Format Options
Window Help
Lotus 1-2-3 Rel. 3
Error: You have attempted to run a protected-mode application in 386 enhanced mode. To run the application, exit and run
Autocad 386
Freelance 3.0
Insufficient memory to run application.
close one or more applications to increase available memory and try again.
OK
Autocad 386
DOS
Lotus
Microsoft Word

Windows 3.0. The multitasking, windowing environment.



DESQview 386 2.3. The multitasking, windowing environment.

Are you still using MS-DOS programs on your PC?

You may want to use DESQview as your primary operating environment.

The new DESQview 2.3 and DESQview 386 2.3 let you use your favorite DOS and DOS-extended programs in windows side-by-side on 80286, 80386 and i486 PCs. As you can see above, you can even run Windows programs within DESQview 386. So the next time you get error messages like the ones at the left in Windows, remember how the same set of programs look running in DESQview.

Whatever standard you use—DOS, extended DOS or Windows—DESQview is still the best way to get the most out of the hardware and software you own today.

DESQview. The obvious choice.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

This comparison was made using a system like the one you might run: Both shots show an ALR FlexCache 33/386 running DOS 3.3 with VGA display adaptor, Novell NetWare v3.01 Rev. A, with IPX/SPX v3.01 Rev A, Microsoft Mouse 7.00, and Microsoft SMARTDrive v3.03 disk cache. Buffers were set to 20. For the Windows screen, we ran Microsoft Windows 3 HIMEM.SYS and EMM386.SYS. For the DESQview screen, we ran QEMM 386 v5.1.

Trademarks: Microsoft, Windows, MS-DOS, 80386, i486, ALR, FlexCache, Novell, NetWare. ©1990 Quarterdeck Office Systems

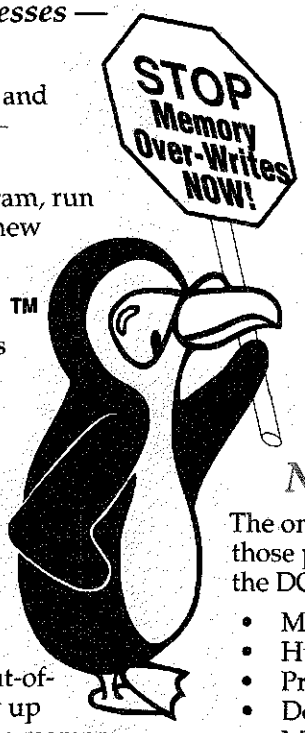
Circle **147** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

POWER DEBUGGING

BOUNDS-CHECKER

Finds out-of-bounds memory accesses — AUTOMATICALLY.



Flush out those Nasty pointer problems and other out-of-bounds memory accesses — AUTOMATICALLY.

Each time you make a change to a program, run BOUNDS-CHECKER while testing the new code. If you accidentally access out-of-bounds memory, BOUNDS-CHECKER will pop up displaying the offending SOURCE LINE. And your program runs at full speed.

"BOUNDS-CHECKER and Soft-ICE make sophisticated use of the most powerful versions of Intel's processor family to track down some of DOS programming's most insidious bugs. If you're developing programs for DOS, these are essential tools."
PC Magazine
July, 1990 pg. 48

Ship Bug-Free Products

You can run BOUNDS-CHECKER while testing your program. There are no additional steps to your testing cycle, but you can feel secure when the program has passed through BOUNDS-CHECKER with no reported problems.

Many over-write problems and other out-of-bounds memory accesses do NOT show up during normal testing. An out-of-bounds memory location may be modified, but that particular location doesn't happen to be important at the time. Once the program is in the field and a certain network is loaded or a certain T&SR or device driver is loaded, that memory location suddenly becomes very important... AND THE SYSTEM CRASHES.

You can prevent these problems by making BOUNDS-CHECKER a standard part of your testing procedure.

Gives you the protection of a protected operating system under MS-DOS.

BOUNDS-CHECKER uses the 386 virtual machine technology to provide real-time memory protection. In addition BOUNDS-CHECKER uses the symbolic information output by your compiler to differentiate CODE and DATA. When your program is running, BOUNDS-CHECKER protects the program's CODE and all memory outside your program.

Requires 386 PC.
MS-DOS is a trademark of Microsoft Corporation.
386 is a registered trademark of Intel Corp.

Soft-ICE 2.5
New Version, New Features

The only debugger specifically designed to solve those problems unique to MS-DOS that we call the DOS Nasties.

- Memory over-writes
- Hung programs
- Program too big to debug
- Debugging T&SRs and Loadable Drivers
- Multiple Symbol Tables
- Supports Microsoft C 6.0 & Turbo C++

Windows Developers

Run CODEVIEW for Windows on a Single Monitor

Nu-Mega has done it again!

CV/1™ $129

Call Today, Have it Tomorrow

BOUNDS-CHECKER	*$249*
Soft-ICE 2.5	*$386*
MagicCV 3.0	*$199*
CV/1™	*$129*

Special Offer...

Buy BC & S-ICE	*Save $100*
Buy S-ICE & MCV	*Save $86*
Buy all three	*Save $186*

30 Day Money-Back Guarantee

Nu-Mega TECHNOLOGIES INC

CALL TODAY (603) 888-2386 or FAX (603) 888-2465

P.O. BOX 7780 ■ NASHUA, NH ■ 03060-7780 ■ U.S.A.

Circle **160** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

January/February 1991 Volume 9.1

FEATURES

GRAPHICS **VGA COLOR CYCLING** *By Michael Abrash* 20
Installment 22, in which Michael explores the complexities of VGA color cycling

GRAPHICS **DIGITAL TYPOGRAPHY** *By Ron Fosner* 32
Typography and font generation for use on graphics screens and laser printers

WINDOWING **WINDOWING FOR TWO** *By Chris Schwartz* 43
A useful Windows 3.0 DLL to access a second monitor just like CodeView

WINDOWING **GUI MADNESS** *By Brett Glass* 46
The trade-offs between character-mode interfaces and GUIs

WINDOWING **TEXT BUFFERS IN WINDOWS** *By Allen Holub* 50
Allen offers the start of a character-based windowing system in Turbo C/C++

DATABASE **WHAT'S ON THE MENU** *By M. Steven Baker* 61
A simple Lotus-style interface for your next database project

C **386 DOS MEETS ANSI C** *By Rex Jaeschke* 67
Rex evaluates three 386 DOS compilers for conformance with ANSI

386 NOW **OPTIMANIA** *By Mark J. Barrenechea* 72
Mark explores code inlining and tail-recursion compiler optimizations

ASM **REAL-MODE VIRTUAL MEMORY** *By John Otken* 76
A real-mode virtual memory system for storage and fast file I/O

OS **ENHANCED KEYBOARDS** *By Mike Schmit* 84
Writing programs to take advantage of extended keyboards

COLUMNS

REDIRECTIONS **AN OPEN LOOK AT MOTIF** *By Frank D. Greco* 10

CONSULTING **NEGOTIATING TRICKS AND TRAPS** *By William H. Roetzheim* 14

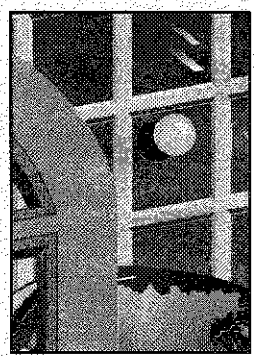
PJ ARMCHAIR **FOR A FEW GOOD BOOKS** *By M. Steven Baker* 87

OFFRAMP **GOLIATH BEATS DAVID—AS USUAL** *By Hal W. Hardenbergh* 96

DEPARTMENTS

EDITOR'S NOTE 4
LETTERS 6
PJ PRESSROOM 89
PJ 1990 ARTICLE INDEX 92
PJ BBS LISTING 90
PJ BILLBOARD 93
ADVERTISER INDEX 95

COVER



Take a pinch of color, add windows, toss in a mouse, and amaze your friends and neighbors with graphics wizardry. This issue is designed to help both casual observers and intrepid programmers see more clearly through those rose-colored glasses.
Artist: Rob Schuster

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P. O. Box 70167, Eugene, OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000293. **POSTMASTER:** Send address changes to *Programmer's Journal*, PO Box 70387, Eugene, OR 97401-0121

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

Depending on your perspective, this past November's elections demonstrate either the cornerstone of our society or an example of the way marketeering wins out over integrity. In any case, it's clear that the *image* of a candidate (intensively displayed on TV) has become all powerful. In the world of programmers, the image of an application is also a matter of central importance.

This issue focuses on graphics and windowing. In the PC hardware market, super-VGA is entrenched. The third-generation super-VGA boards, with a megabyte or more of video memory (1024x768 in 256 colors) and fast memory systems, will take the original IBM VGA design well into the future. At the high end, the more intelligent TI TMS34010/34020 boards (32-bit microprocessors in their own right) will continue to make inroads. Even before the IBM 8514/A and its clones had established a solid market base, IBM introduced the XGA (surprisingly, IBM released specifications this time). Nevertheless, the EDSUN RAM/DAC chip is what stole the show at the recent Comdex. This inexpensive chip replacement to stock SVGA boards provides anti-aliasing and allows the display of up to 740,000 colors with a 256-color VGA board.

WINDOW WASHING IS OVERDUE—THE IMAGE IS THE MESSAGE

The problem plaguing graphics and user-interface programming is the plethora of software choices. The hot item these days is Microsoft Windows 3.0. Part of the fallout from the IBM and Micrsosoft OS/2 split is that Microsoft moved a couple of hundred people from OS/2 to Windows development. Even though a number of these folks were "marketing types," a transfer of this size will undoubtedly make a difference in the future of Windows.

From a technical perspective, I'm not a fan of Windows 3. This third time around Microsoft had the chance to do some things right and didn't go far enough in a number of areas. If you've worked on a Mac, you know that the ability to click on a document or data file and not on the application makes a dramatic difference. Although the Apple file system facilitates this process, some special Windows EXE file headers and hidden files could have extended this concept to the DOS environment. Windows 3 also incorporates some fault-ridden features. To DOS, it appears to be one huge application, so SHARE and other file protection mechanisms are useless. Yet Windows provides no equivalent internal file sharing protection for its applications—a single app or DOS window is free to trash the files of another. Windows 3 is really three distinct products in one—Windows, Windows/286, and Windows/386. Remember, packaging and image reign supreme. So now the choice of which base to target an application to is left up to the developer.

AND SO IT GOES ... ON TO THE 80586

What Windows most noticably lacks is true 32-bit support for the 386/486. OS/2 2.0 was supposed to provide this; now that IBM has assumed responsibility for most OS/2 development, the market is even less assured. Fortunately, 386 DOS extenders can use DOS Protected Mode Interface (DPMI) to supply Windows 3 compatibility.

This makes DOS extender products even more entrenched in the market. The 386 DOS developer has new tools from MetaWare, Watcom, and MicroWay, with symbolic debuggers that rival CodeView. MicroWay, MetaWare, and Ergo have licensed the same windowing debugger code, so look for similar user interfaces. Phar Lap will soon be releasing an interesting 286 DOS extender that runs (non-PM) OS/2 executables—it supports the OS/2 1.1 API. At Comdex, I saw MicroWay's impressive i860 coprocessor board running at half the speed of a CRAY-1. So 1991 will see a raft of 386-clone chips, 50MHz Intel 80486s, and early 80586 chips. And though image may appear to prevail, I expect speed will continue to be an issue of increasing importance in the months ahead.

Steven Baker

Programmer's Journal

Editor
M. Steven Baker
Associate Editors
Cecelia Hagen
Deborah Healey
Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan
Founder
Greg Estes

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Subscriptions
Pamela Coleman
Office Staff
Jill Edwards
Barbara Curtright
Sylvia Long
Randy Randolph
Production & Art Director
Wanda Walker
Production Artist
Nicole Harris
Illustrations
Jim Carpenter

Sales & Marketing Director
Robert M. Tappan
Advertising Manager
Jay Moore

Business and Editorial Offices
Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
(503) 747-0800
(800) 234-0386
Fax: (503) 746-0071

Subscriptions
1 year (6 issues) $24.00

Change of Address
Programmer's Journal
PO Box 70387
Eugene, OR 97401-0121

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OCR LIBRARY

Join the scanning and imaging revolution. Add ***Omnifont Optical Character Recognition*** of printed or typed text directly into your DOS or Windows 3 application with OCR\Library™. OCR\Library is available in .LIB or .DLL formats for most DOS and Windows compilers.

WORLD'S LEADING TECHNOLOGY

A leading team of ***Soviet mathematicians*** working in the United States with top U.S. software engineers developed OCR\Library. Revolutionary new algorithms have been employed in a feature based approach to recognizing scanned text of virtually any font and size. The flexible library approach allows developers to customize the leading OCR for their user interfaces and applications.

COMPLETE OCR RECOGNITION

- recognizes font sizes 8 - 72 points, bold, underline, etc.
- recognizes all non-stylized fonts including dot matrix
- separates pictures from text & recognizes text placement
- has device independent scanner and bitmap interfaces
- employs optional 100,000 word spelling dictionary
- maintain dynamic spelling dictionaries
- allows flexible zone definition on pages
- includes optional foreign language character sets
- can learn to recognize special symbols

IDEAL FOR APPLICATIONS IN ...

- image databases
- desktop publishing
- word processing
- accounting
- hardware bundling
- personal & corporate databases
- language translators
- synthetic voice applications
- document processing
- fax recognition applications

The speed, recognition accuracy, and memory size of OCR\Library set a new standard in OCR technology.

*OCR\Library will be available soon for OEM licencing at reasonable prices. **Now is the time** to plan the integration of OCR\Library into your application.*

COLEMAN SOFTLABS INC

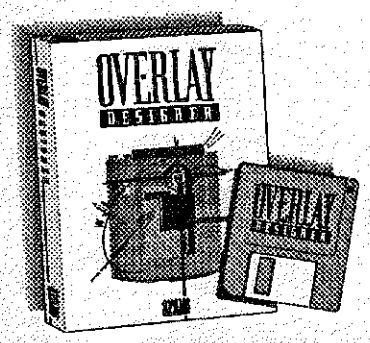
Coleman Softlabs Inc.,
296 Bay Road,
Atherton, CA 94027.
(415) 322-9006 phone
(415) 322-1622 fax

Circle 88 on Reader Service Card

OVERLAY DESIGNER™

for use with

Plink86 ® and RTLink ®



IF YOUR PROJECT BOGS DOWN

in the tedious detail work involved in designing efficient overlay structures, Overlay Designer is the software tool to save you time and keep your project on schedule.

DO NOT BE MISLED ...

by products that claim to do overlays automatically for large DOS applications. Important tradeoffs between size and performance still must be made by programmers who understand the code and the user interface. The powerful ***"what you see is what you linked"*** interactive Overlay Designer editor, tabulation, and graphical modeling program is the only workable solution. Early adopters of Overlay Designer use it regularly and find it an ***indispensible tool for overlay design.*** Context sensitive help and automatic .MAP and .OBJ file reading give immediate results with ***virtually no set-up required.***

"After my first day with Overlay Designer, I had done work that would have otherwise taken me 2 weeks."
- Mike Milsner, Senior Programmer
Symantec Corporation

Overlay Designer is backed by a 30 day unconditional money back guarantee.

COLEMAN SOFTLABS INC

Coleman Softlabs Inc.,
296 Bay Road,
Atherton, CA 94027.
(415) 322-9006 phone
(415) 322-1622 fax

Circle 39 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

CRIMINAL COMPILERS

I have read *Programmer's Journal* for a few years now and consider it to be one of the better programming magazines. But now it's time for me to get my two cents' worth in on the "mine is better than yours" language issue that crops up from time to time.

I find that more of the debate is fueled by inadequate compilers than by actual language differences. Much of the defection to C is the result of programmers being fed up with languages whose compilers generate anything but optimized code. At least C allows the programmer to do the optimizing; unfortunately, it also *requires* that it be done. Though the result of this may be reasonable utilities, it also means lousy compilers. If the programmer is so willing to do his/her own optimizing, why should the developers spend any of their time trying to make the compiler smarter?

In developing the Masterful Disassembler (MD86), I had many occasions to review code produced by an assortment of compilers. While some of it is adequate, a lot of it really stinks. I use Turbo Pascal for most of my programming and, in general, I like it. However, this compiler (v5.5) generates irritatingly bad code at times. For example, the statements:

```
i:=LastValue;
if  i > MaxValue then MaxValue:=i;
```

are turned into the following assembly language code:

```
    MOV   AX, [L0040H]
    MOV   [L003EH],AX
    MOV   AX,[L003EH]
    CMP   [L0042H],AX
    JLE   1$
    MOV   AX,[L003EH]
    MOV   [L0042H],AX
1$ ...
```

I'd like to strangle the guy who wrote the code generator here! It's hard to imagine less efficient code.

Now maybe I could write this a different way to help the compiler out, but why should I? Why should programmers have to make up for the inadequacies of compilers?

Although many people look down on FORTRAN and COBOL for being "old fashioned" mainframe languages, at least they had good compilers. Maybe if we all rant and rave enough and expose these code-wasting compilers, the developers would see fit to improve them. God knows there's plenty of room for it!

Clark A. Calkins
Walnut Creek, CA

ILLUSTRATION BY JIM CARPENTER

LAHEY ENHANCES FORTRAN

Thank you for your recent coverage of 32-bit FORTRAN and DOS extenders ("FORTRAN on the 386," *PJ* 8.4). As you know, the popularity of 386/486 personal computers has led to an increase in the use of FORTRAN on microcomputers.

In the benchmark results, F77L-EM/32 legitimately diagnosed an underflow in the Superlite program. Since your review appeared, we have added Weitek support for our arithmetic exception subroutine, UNDER0(*lflag*). When called with *lflag* set to **.TRUE.**, a store-into-memory operation will store a zero value instead of reporting the underflow. In Superlite, inserting

```
CALL UNDER0(.TRUE.)
```

before calling the **CONFAC** subroutine will prevent the failure. We produced a runtime of 82.72 running on a 20MHz 386 with the Weitek coprocessor.

Guy Ceragioli
Lahey Computer Systems Inc.
Incline Village, NV

UART CURIOUS

I am very late in my reading, so I have just gotten around to Steven Baker's "PC Communications and Protocols" in *PJ* 7.4. Since I don't really understand the hardware and since I don't have (or want) access to the National Semiconductor documentation, your discussion of UART's in the article tantalized me.

Do you know of a public domain program which will tell me which UART is on COM1 and which is on COM2? Is there some sample ASM or C source which shows how to turn the buffer of the NS16550 on and off? For that matter, is there some article for those of us who are ignorant about hardware explaining the NS16550's buffer on a programmer's level?

Lewis Paper
Roseville, MN

THE EDITOR RESPONDS: National Semiconductor provides a free bulletin board system that offers sample source code to test for various PC UART chips and enable the buffer in the NS16550. The bulletin board can be reached at 408-245-0671. Their literature department has databooks on the NS16550 and the other available PC UART chips, which contain both chip descriptions and sample code.

ORDER YOUR ORDER HERE

John Otken has done a terrific job of presenting good advice and useful techniques in his "Manipulating Large Data" (*PJ* 8.2). I hope that he keeps writing for your magazine. I would like to comment on ORDER, a program to resequence numbered local labels in an assembly-language program which accompanied Otken's article.

First, lest my comment be taken negatively, let me stave off *ad hominem* defensiveness by confessing that I have written my share of assembly programs, both stand-alone and interfaced with "higher level" languages. It's fun. It is an art and it is a science. But assembler, like any other programming language, is a tool. I suggest that there are much better tools for building the ORDER program. Otken's description of the program cries out for an implementation in AWK or SNOBOL. Here are some clues: "The program ... identifies a procedure by a line that begin[s] with one or more alphanumeric characters followed by blank space and the PROC keyword ... copies unprocessed source lines ... label table ... searches for a local label formed from three letters and some digits ... scans each line ... searches for the index in the **label_table** and substitutes the new index ...."

AWK primitives perform all of the operations mentioned in this eclectic quotation from the article. Writing ORDER in AWK takes less time, less effort, is much more maintainable and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

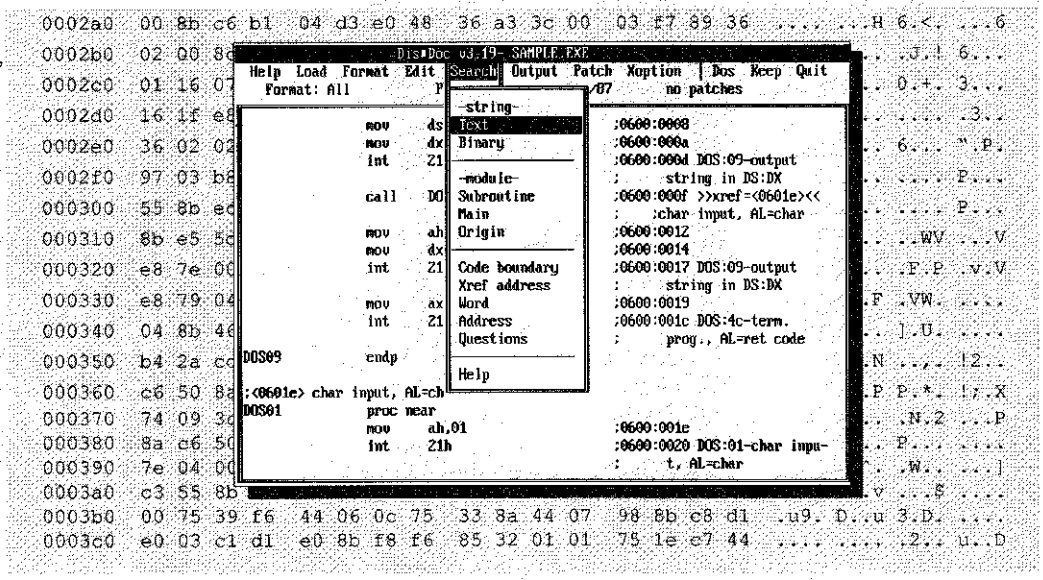
NEW VERSION

No Source?
No Problem.

Automatic BIOS, PSP, MS-DOS interrupt labeling built-in

Batch and Interactive modes

Interactively locates code/data boundaries



8086 to 80486 instruction sets supported

Interactive cross-referencing

Creates listing only after disassembly is perfect

Dis • Doc Professional™

New **Dis • Doc** *Professional* is your **dual-mode** key to any DOS source code. It works in batch and interactive modes simultaneously, allowing you to generate the core information of even the most complex programs **fast**...and modify them even faster. Most programs will come apart in just two minutes. Imagine what you can do with a tool this powerful! **Dis • Doc** sifts through programs **eight times** for guaranteed accuracy. When code gets mixed up with data, our toolbox comes to the rescue with **smart** search and **easy** edit utilities.

Warning: Dis • Doc *Professional* may change the way you work **forever**.

Programmers who used to shy away from fixing outmoded programs with no source code are going to discover a **valuable** new talent: the ability to modify and revise codes that would cost way too much to start over (it's a programming manager's dream). Save your employer huge new-programming fees and enhance your marketability. **Dis • Doc** is so easy to learn, you'll be a high-dollar hero in no time!

Dis • Doc *Professional* is an amazing new teaching tool. Learn how programs work...take them apart and see the writing techniques that top pros use. Use it to assist in debugging. Hunt down viruses and write killers. **Dis • Doc** can save you **years** of frustration, and it only costs $249.95.

COMPARE FOR YOURSELF!

	Dis•Doc™	Sourcer™
Speed of disassembly (bytes/sec)	***822***	*120*
Accuracy (code/data separation*)	***99.9%***	*99.9%*
Unlimited file size	***YES***	*NO*
BIOS labeling	***YES***	*XTRA $$*
BIOS/PSP file labeling	***YES***	*NO*
Unpacks EXE files	***YES***	*XTRA $$*
Patcher	***YES***	*NO*
Batch mode	***YES***	*YES*
Interactive mode	***YES***	*NO*
Time it takes to customize a label	***5 sec.***	*2-5 min.*
Time it takes to change data to code	***5 sec.***	*2-5 min.*

*at eight passes based on a 9040 byte EXE file picked at random

To order your **Dis • Doc Professional™** disassembler toolkit or our $3 demo disk, simply call:

1-800-336-1961 in US
1-203-953-0236 outside US
or send check or money order for **$249.95** plus $5 s&h to:

RJSwantek, Inc.
178 Brookside Road
Newington, CT 06111-1320 USA

For more information call 1-203-953-0236 • FAX available • MasterCard & Visa • Shipped Immediately Via UPS Blue inside US.

Sourcer is a registered trademark of V Communications, Inc.

Circle **59** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

readable, and overcomes deficiencies built into the assembly language implementation of ORDER ("the assembler can differentiate among these labels, but ORDER cannot").

How many of you assembler gurus out there could write, test, debug, and then eliminate ORDER's deficiencies in three hours or less? A novice at AWK did it in two. The AWK implementation surely is slower than the assembler version, probably by an order of magnitude. It rumbles through 100 to 500 lines per second on the 386 machine, depending on the number of labels to fix up. This will cost about a second each time you use it on a modest source file. But stack that up against the 50,000 or so extra seconds required to code the program in assembly. Keep in mind the fact that ORDER.AWK can handle large and complex files, far exceeding the 64K file-size limit of ORDER.ASM. And don't forget the possibility of changing aspects of the algorithm.

The moral to draw here is that every task has its tool. When you build a ship in a bottle, you use tweezers and a scalpel. Houses, too, can be built with tweezers and scalpels. But when you build houses for a living, you use a hammer and saws. ORDER is a house, AWK is a hammer, assembler a scalpel.

William A. Huber
Philadelphia, Pennsylvania

ED NOTE: ORDER.AWK is available on this issue's code disk.

HOW UNIVERSAL ARE THEY?

I read Brett Glass's article with interest. Having surveyed over 40 different data bases (relational and flat file) in the past six months, I would like to offer one comment.

For over two years I have prayed for a product with the ability to read/write to varied databases without the need for import/export operations. Software firms want to develop products that have multiple database engines to increase their customer base. But their "universal" toolkits have not turned out to be as much of a dream as I thought.

I recently found out about Advanced Revelation by Revelation Technologies. While I have not used it in practical applications, I have reviewed the demo and associated literature. Apparently, a programmer can write the code and apply it to any data base. You simply need to identify the data base in a dictionary with the type, etc. If the format is not known to AREV, you must explain the file format; from then on it will read/write that type of file. They call this "environmental bonding."

It could be the answer to our prayers; it certainly is worth investigating. I am also sure that other readers would be very interested in the concept of "environmental bonding" for their own database applications.

Michael V. Vaccarello
Dallas, PA

INSTALL.BAT
First impressions last.
INSTALL 3.0 gives your software product a professional introduction.
Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.
INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.
•Features very sophisticated error-handling
•Does high performance data compression
•Can safely and reliably modify system files
•Allows selective installation of parts of your program
•Complete source code included
•No royalty charges
•Free technical support
INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.
INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.
KNOWLEDGE DYNAMICS CORP.
Highway Contract 4, Box 185-H
Canyon Lake, TX 78133-3508
MasterCard/VISA/COD/POs welcome
$249.95 + shipping
Money back guarantee
Sales 800-331-2783 Ext 0196
Internat'l 512-964-3994
24 hr FAX 512-964-3958
24 hr BBS 512-964-3929
Circle 295 on Reader Service Card

The Programmer's Corner
— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today (300-9600 Baud): (301) 596-1180
(301) 997-3744 (301) 621-4636

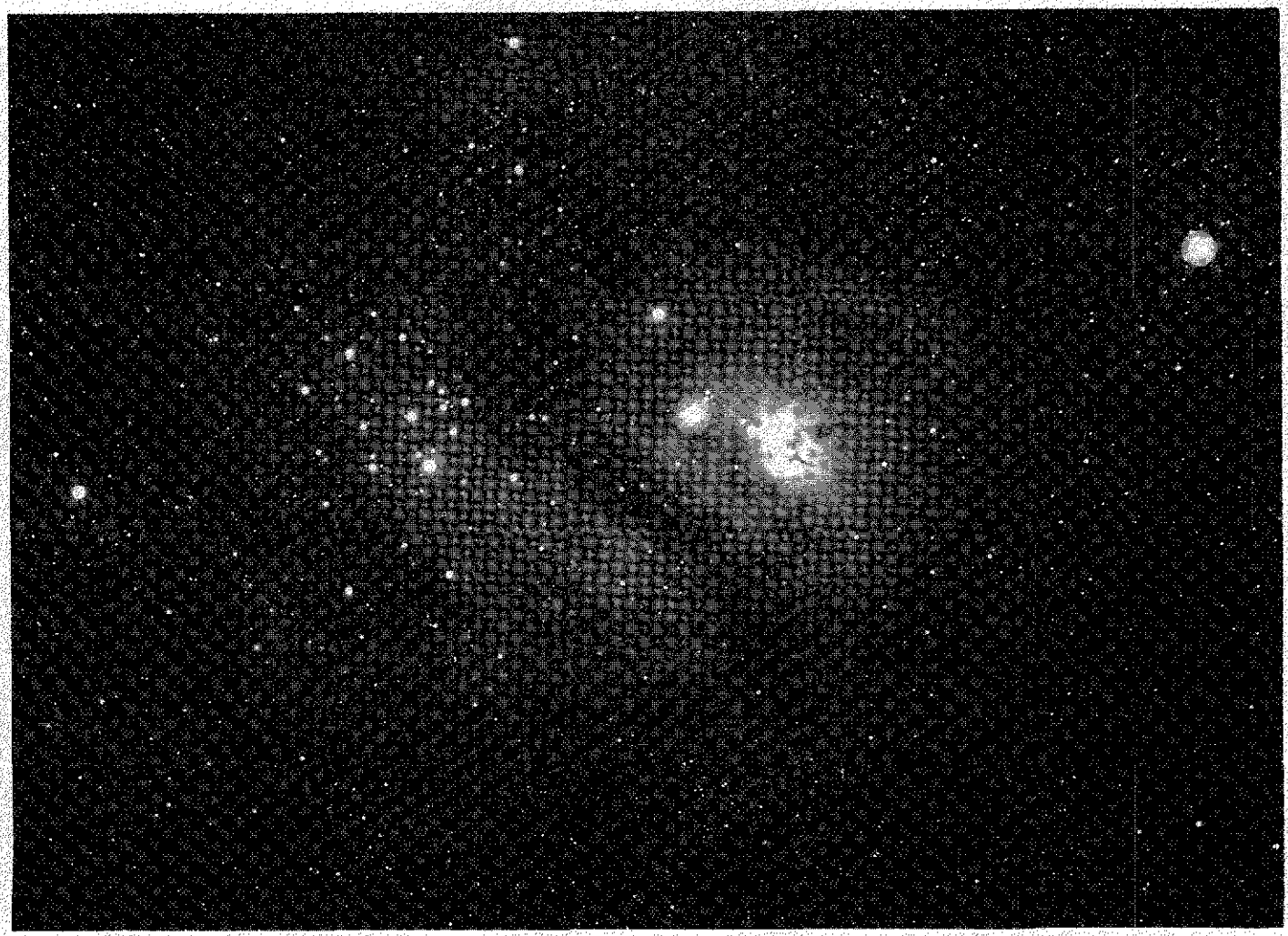
Only $50 for a one year subscription

— TPC uses FastComm 9600 V.32 Modems —

TPC
The Programmer's Corner
5430 F Lynx Lane, Suite 224
Columbia, MD 21044

Circle 54 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Lagoon Nebula, a 50-light-year-wide cloud of dust and gas, 4,500 light-years from Earth. National Optical Astronomy Observatories, Kitt Peak National Observatory. Resource: "The Universe and Beyond," Terence Dickinson, Camden House Publishing.

Travel the Universe Beyond 640K

.RTLink®/Plus 4.1 with Virtual Memory Linking

For Microsoft® C, QuickC®, MASM, Fortran, Pascal, CodeView® and Clipper® S'87/5.0

The .RTLink/Plus linker provides memory management tools which allow MS-DOS® programs larger than 640K to execute. However, .RTLink/Plus is not just for large programs. When LAN requirements, Multitaskers, TSRs, etc., squeeze your program out of conventional memory, .RTLink/Plus allows execution in the space available.

.RTLink/Plus provides a complete set of Memory Management/Linker tools: VML™ (Virtual Memory Linking); Multiple and Nested Overlay capability; and VML combined with overlays. VML will get your program running without overlays or other hassles. If fine tuning is required for your particular application, .RTLink/Plus allows simple or detailed adjustments to be made.

Virtualized programs automatically take advantage of the target machine's memory environment. Programs will execute in conventional or expanded memory (if present). Virtual 'pages' or overlays will automatically cache intelligently using conventional memory or disk...or from expanded or extended memory, if present.

.RTLink/Plus provides a software solution for DOS memory problems which is valid for XT, AT, 386 and above.

Data specific to a routine may be placed in pages freeing up more conventional memory.

The unique *Dynamic Data Page* feature allows changing data to be placed in virtual pages.

No source code changes are required.

A high quality, built-in CPU profiler is provided at no additional charge.

For additional information

WRITE: Pocket Soft, Inc., 4.1 Info Pack, P.O. Box 821049, Houston, TX 77282, USA

FAX: 713-460-2651 CALL: 713-460-5600

Pocket Soft, Inc.

P.O. Box 821049 • Houston, Texas 77282

Pocket Soft is an authorized licensee of Microsoft CodeView Information. "Microsoft," "MS," "QuickC," "MS-DOS," and "CodeView" are registered trademarks of Microsoft Corporation. .RTLink is registered trademark of Pocket Soft, Inc. VML and Pocket Soft are trademarks of Pocket Soft, Inc. Clipper is a registered trademark of Nantucket Corporation.

Circle 279 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

An Open Look at Motif

By Frank D. Greco

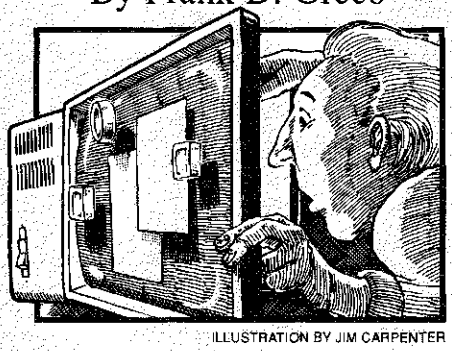


ILLUSTRATION BY JIM CARPENTER

As is typical for the powerful and flexible UNIX operating system, there are several window systems and user interfaces to choose from. The most popular window system in this neck of the woods is currently the network-based X Window System, which people frequently but erroneously call "X Windows." The X Window System is not—I repeat, is not—a user interface: it is a machine-independent system that programmers can use to create, modify, manage, and destroy sections of screen real-estate (i.e., "windows").

FREE PARC-ING

In yet another example of their unintentional altruism, Xerox PARC researchers came up with one of the most important developments in computer-human interaction. Besides Ethernet, Xerox PARC originated the mouse, the PostScript page description language, remote procedure calls, multi-threaded processing, the concepts of window systems and graphical user interfaces, object-oriented technology, and arguably even the first personal computer. Many of these landmark Xerox innovations date back to the ancient 1970's.

The X Window System was originally designed at MIT for two reasons: there was a need for a multi-process debugger, and MIT's Project Athena needed a generic window system for hundreds of workstations to be donated by IBM and DEC. Since X draws on some of the seminal Xerox work, it is yet another result of Xerox's legacy.

Unlike Microsoft Windows, the X Window System is based on the client-server programming model which communicates via a simple, network packet protocol called the X Protocol. To ease programming, the MIT designers created a low-level API called Xlib.

SOME PANE, SOME GAIN

In the realms of DOS and UNIX (there isn't an OS/2 realm yet, and probably never will be) there are several successful window systems. Serving as both a window system and a user interface ("a floor wax and a mouth wash"), Microsoft's Windows is the reigning king in the DOS/80X86 community, especially with all the hoopla surrounding Windows 3.0. Using MS Windows is fine as long as you have an 80X86-based computer—since MS Windows *only runs* on an 80X86-based computer.

You may hear from the uninitiated that the windowing people have client-server networking terminology backwards. Neophyte network hackers say that the servers are "out there on the network" and the clients are the workstations. This is not always true. Recall that a server is a process that accepts requests for services, which are typically initiated by client processes. For example, a "file" server is a process which accepts file I/O requests from "file" clients, attempts to satisfy the requests via intimate negotiation with a disk drive, and then returns the requested disk blocks to the "file" client.

Usually the "file" server is a physically large machine because the disk drives are enclosed in the machine's cabinet or in close proximity to the computer. The "file" clients, however, are typically small machines because they don't have to have disk drives: all they need to have is a network connection.

WINDOW SERVERS AT WORK

In a similar fashion, a "window" server is a process that accepts windowing requests from "window" client processes. These clients, just like "file" clients, may reside anywhere on the network. A "window" server may honor any of its remote clients' requests by, for example, drawing a rectangle on the display, popping up a menu, or changing the mouse cursor for an application. A heuristic here is that there is one "window" server per physical display.

If you have a slow machine connected to the network and want faster performance, you can simply opt to run your programs (i.e., X clients) on faster machines on the network and have your mouse, keyboard, and graphics I/O logically connected to them via the X Window server. In a way, the concept of the X Window server is just like the old mainframe-terminal model, except that now the "terminals" are highly intelligent, and graphics and window manipulation are very portable. In fact, it's so similar that some vendors (e.g., NCD, Graphon, and Wyse) are attempting to create a market devoted to X terminals, relatively inexpensive machines whose sole purpose in life is to be X Window servers (in this model, X clients MUST be run remotely).

NOT AN X-DICTATOR

The X Window system was purposely designed not to dictate any user interface. Its architecture lets programmers stack additional layers of software for a specific look and feel. The MIT developers also built what they hoped would be a programmers' toolkit called the X toolkit (Xt) Intrinsics. This quasi-object-oriented layer is what the MIT folks suggest software developers use to implement a particular look and feel (or, as the MIT documentation says, appearance and behavior).

ECONOMICS 01100101

In X parlance, you use the Xt Intrinsics to build user interface "widgets" (an unimaginative, unrevealing and confusing moniker). Widgets are created to interact with the user. Due to the architecture of X, Xt-based widgets can extract a high toll on performance. So X developers at HP created higher-performance graphical objects they called "gadgets" (how's that for paucity of imagination!),which are lighter-weight (i.e., with less context) widgets.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Language

AI Expert Systems
ARITY Combination Pkg 974
Exsys Professional 694
PC Scheme LISP by TI 74
Turbo Prolog Toolbox 70
TURBO PROLOG V2.0 110
VP Expert 219

Assemblers
Advantage Disassembler 279
ASM Flow 89
MS Macro Asm 97
OPTASM 129
Re:Source 129
Turbo Assembler/Debugger 98
Visible Computer 80286 84

Basic & Addons
DB/Lib 120
MS Basic 7.0 339
MS QuickBASIC V4.5 69
Probas by Hammerly ..... call
ProScreen by Hammerly ..... call
QBase 124
QuickPak Professional 139
Softcode 69
True Basic 69
Turbo Basic 69

C LanguageCompilers
High C 386 875
Lattice CV6 189
Microsoft QuickC w/QASM 135
MSC 6.0 345
Top Speed C Std 179
Turbo Cby Borland 97
Turbo C Professional 165
Watcom C8.0 386 std. 719

Cobol
MS Cobol V3.0 629
Microfocus COBOL W/tools 1499
Realia COBOL 844

Fortran
For_C FORTRAN'77 to C 670
Lahey FORTRAN F77L 525
Lahey Personal FORTRAN 84
MS FORTRAN with Codeview 295
RM Fortran 494

Modula2
Logitech Dev. Sys 195
Topspeed 3 pak 159

Object Oriented Programming
ACTOR 559
Whitewater Resource Tool Kit 160
Smalltalk/V 286 169
Turbo C++ 109
Turbo C++ Prof 159
Turbo Debugger & Tools 99
Zortech C++ with source 205
Zortech Dev't Kit 380

Other Languages
IntegrAda 724
Janus Ada 369
Meridian Ada Dev't Kit 980
MuLISP87 219
PC/Forth+ 199
Personal Rexx 129

Turbo Pascal
Turbo ASYNCH PLUS 115
Turbo Pascal 5.0 105
Turbo Pascal 5.0 Professional 175
Turbo POWER TOOLS PLUS 124
Turbo Programmer 444

DBMS

Data Base Development
DBXL 199
Clarion Prof 525
Clipper S87 399
dBASE III Plus 450
Force 395
FoxBASE + ... 239
Clipper 5.0 495
Fox Pro Lan 745
Fox Pro SU 579
QuickSilver 379

dBASE Library
Artful 189
CodeBASE IV 215
Funky 175
IDL 125
Netlib 229
Proclip V3.5 119

File Management
Btrieve 180
Cindex Plus 329
CISAM 325
CQL wSRC 349
WPASS 349
ctree ..... call
dBC III PLUS 435
db_FILE RETRIEVE 265
Essential BTree w/source 149
FABS Plus 172
Netware C interf. 239
Netware SQL 489
pBase 185
Turbo Programmer/C 449
WKS Lib 189
XDBC 555
XQL 599
Xtrieve PLUS 459
Btree Filer 97

Addons

C Communications
Newt C Async Lib. sy Silverware ..209
BreakOut II 99
C ASYNCH MANAGER 155
Essential Communications 295
Greenleaf Comm Library 279
Greenleaf ViewComm 315
Lattice Comm. Library 179

Dbase Addons
AdComm 99 105
BLINK 179
CLEAR + FOR dBASE 139
Dbase Online V2.0 125
dBugsource debugger 170
dBXdBASE III to C 460
dClip 189
DCS Report Gen 265
desQ Clipper Api 469
dGEbusiness graphics 259
Documentor 225
dQuery 165
dSalvage 79
FLIPPER Graphics Library 175
Spellcode 89
Touch & Go 325
R&R Code Generator 125
R&R Relational Reportwriter 125
Scrimagescreen/menu 129
SilverComm Library V2 169
Silver Paint Library 79
Steve Straly's Tools 169
UI ProgrammerDev's Version 2.0 ..419

General Addons
C Tools PlusV5.0 105
C Utilitiesby Essential 154
Greenleaf Functions 159
Greenleaf SuperFunctions 209

Graphic Addons
Essential Graphics 345
Graphic C 329
GraphicsMenu 159
Greenleaf MakeformDOS 90
GSS Dev't Toolkit 505
Halo '88140 + dvices 275
Menuet 259
MetaWINDOW/PLUS 269
Mewel 149
PCX Programmer's Toolkit 175
Slate w/Graphics 415

Text Screen Addons
AEWindows 204
C Worthy w/forms 259
Cursesby Aspen Scientific 125
Greenleaf DataWindows 325
HiScreen XL 124
HAM by JYACC 464
VC Screenpainter 114
Vitamin Csource, menus 164

Utilities
BACKIT 120
COPY II PC 35
Disk Technician Advanced 149
FASTBACK Plus 142
MACE GOLD 129
MACE Utilities 90
Norton Utilities Advanced 101
PC/Tools Delixe 6.0 95
V opt 111
XENOCOPYPC 70
XTree Pro 111

Miscellaneous

Other Products
Aspen Korn Shell 109
CO/SESSION 214
DESQ 386 ..... call
dQuery/Lib 274
Line & Locate ++-ROM MSC. . 339
Printware 149
Smartem 240 294
Stingray Clarion Lems ..... call
QEMM 386 55
386 Maxpro 115

Case & Prototypes
Demo II 155
NEW! Easy Case + ... 205
Instant ReplayNostradamus ... 134
Interactive EasyFlow 135
MetaDesign by Meta Software 329
ProC 425
PROTEUS Prototype System . 124
Show Partner F/Xdemos 315
Softdemo 59
Protoview 499

Tools

Debuggers
386 DEBUG 155
MultiScope 135
Periscope OK 465
Softprobe 86/PX 345

Development Tools
DASM 239
MKS AWK 84
MKS Lex & Yacc 205
MKS RCS 155
PC Lint ..... call
Plink 86 Plus overlays 399
PolyMake 129
PVCS Corporate 355
.RTLINK + by Pocket Soft 409
TLib 89

Editors
BRIEF ..... call
Epsilon 155
KEDIT 119
Me wsource 169
MKS VI 125
MultiEdit 90

★ SAGE PROFESSIONAL EDITOR ★
Mouse Support, DOS/OS/2,Emulates Brief, vi & Emacs, EMS Support Fully ProgrammableList 295 Ours 249
30day Money Back Guarantee

NROFF/PC 117
PC/EDIT+ 269
Qedit 79
SPF/PC 189

Windows & OS/2

Windows
Asymetrix Toolbook ..... call
Case:W 754
CTalk/views 375
Dbfast/Windows 325
DialogCoder 429
MS Windows SDK 325
MultiScope Windows 285
Object Graphics 319
ProtoView 619
RF Flow 69
WinTrieve 335
WNDX Gui Tools 445
Windows 3.0 99

OS/2
Brief ..... call
Btrieve 475
C Tree 309
C Worthy 339
Greenleaf Data Windows 319
Mks Awk 159
Mks Lex & Yacc 339
Mks Toolkit 399
Multiscope 340
OS/2 PM Toolkit 359
VC Screen 139
Vitamin C 289
PC Yacc 620
Zortech OS/2 129

1-800-323-1809

PROGRAMMERS WAREHOUSE

7819 East Greenway Suite 8 • Scottsdale, AZ 85260 • Outside US call (602) 443-0580

All Prices and Offers Subject to Change Without Notice

PROVIDED AS IS BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Some vendors (such as Sun, Solbourne, and Neuron Data) feel that the Xt API is clumsy and slow (and I agree). These vendors have developed their own X Window toolkits based on the Xlib layer. Sun has their XView OPEN LOOK toolkit (the source code is free), the neural network specialists Neuron Data offer the multiple look-and-feel OIT toolkit, and Solbourne offers the OI toolkit based on C++ (which also sports a multiple look-and-feel design).

THE MOTIVE BEHIND MOTIF

Two main user interfaces are currently battling it out for prominence in the workstation community: OPEN LOOK and Motif. (More about OPEN LOOK in a minute.) Motif is a graphical user interface combining the Motif toolkit based on the Xt Intrinsics ("the Motif Widget Set"), a presentation language called User Interface Language (UIL), a window manager (Motif Window Manager—MWM), and the Motif style guide. As far as I know, there is only one toolkit, the Motif Toolkit, that generates the Motif look and feel.

Motif was developed by what I believe must have been some form of "contrived consensus." Motif is a "product" by the not-for-profit Open Systems Foundation (OSF) organization. OSF was formed by companies that believed in the definitely-for-profit Closed Systems. Motif is the result of a combination of software efforts and monies donated by OSF-founders DEC, HP, and IBM.

According to the OSF, Motif was chosen from a pool of many other user interfaces purely on the basis of its technical merit. This is about as believable as Dan Quayle's winning at Jeopardy. It's no surprise that out of all the user interfaces submitted to the OSF (of which OPEN LOOK was one), Motif came out ahead. It must have been like competing for a job with someone whose father just donated a million dollars to the company during a golf game with its chairman.

HUI, DUI AND GUI

The OPEN LOOK user interface was developed by Sun Microsystems (with some help from AT&T), and its makers freely acknowledge the pioneering work done by Xerox. As a matter of fact, some people at Xerox contributed significantly to the OPEN LOOK design effort. As opposed to Motif, OPEN LOOK is a specification, not a piece of software. This is an important distinction. You can implement OPEN LOOK using several different toolkits on several different window systems. Two popular OPEN LOOK toolkits are the Xt-based OLIT (previously known as Xt+ or Xol) from AT&T and Sun's Xlib-based Xview toolkit. (Xview is a higher-level toolkit based on a lower-level X Window layer ... are you following this?)

There are also OPEN LOOK toolkits based on the innovative PostScript-based NeWS window system (for example, The NeWS Toolkit TNT). Some vendors have even announced OPEN LOOK for MS-DOS and the Macintosh. Since OPEN LOOK is a specification and not software, programmers can implement the OPEN LOOK user interface via several different APIs.

Instead of Motif, I prefer the elegance of the OPEN LOOK user interface, which I believe will be the user interface of choice for a vast share of the upcoming boom in sales of commodity workstations. There are several reasons why:

- The majority of these workstations will be Sun-flavored and SPARC-powered (nope, they won't be 486-based, Hal). There will be a SPARC clone market which will be as hotly contested as the PC clone market. All the clone vendors will follow Sun's lead, just as most PC clone vendors follow IBM's lead. Sun will NEVER support Motif: it represents political anathema to Sun Microsystems.
- OPEN LOOK software comes already bundled with these workstations. Programmers and users, quite naturally, use what comes with the machine.
- There will be a plethora of shrink-wrapped OPEN LOOK applications. The ratio of available OPEN LOOK applications to available Motif applications is already 3-to-1. Motif will be as popular on workstations as Digital Research's GEM environment is on PCs.

WINDOW MANAGERS

Lately I have been involved in many meetings with my clients regarding user interfaces and SPARC workstations. Most of the people who say they *must* use the Motif user interface are PC-minded corporate managers who are now buying SPARC-based machines for their power users and applications developers. These managers believe that just by choosing the Motif user interface, their portability problems will disappear—because Motif *looks* like Microsoft Windows or Presentation Manager. Ouch!

SEE YOU IN COURT?

Now that the Motif style guide (or user interface specification) is in the public domain, and it's well-established that Motif borrowed its look from Microsoft Windows and Presentation Manager, will the specifications to Microsoft Windows also be in the public domain? If a vendor came up with a more efficient manner of windowing on PCs than Microsoft and it matched the look and feel of MS Windows or Presentation Manager, what legal right would Bill Gates have to the ownership of the look and feel of MS Windows? PJ

Complete Modem Communications

TURBO PASCAL & C programmers

Introducing

Sugar Coated Serial™

A complete library of Turbo Pascal & C procedures, functions and low-level assembly routines that allows you to EASILY incorporate MODEM communications into your application.

Features:

- All source code included (including Assembler).
- No knowledge of assembler required.
- No royalties or fees for use in your programs.
- Easily do XMODEM, YMODEM, & Telink file transfers (including multiple files with YMODEM.)
- Pull-down menu communications program included using only routines from this library.
- Complete user manual including many examples.
- Much more...call for the information pak to get the full details.

For IBM PC/XT/AT compatibles using T.P v5.0 or >

* Complete System: $109.95
* Demo Disk: $5.95
* Information Pak: Free

Shipping included, IL residents add 6.25% sales tax. Specify disk size.

M/C Visa

Send Check, M.O. or Credit Card info with order to:

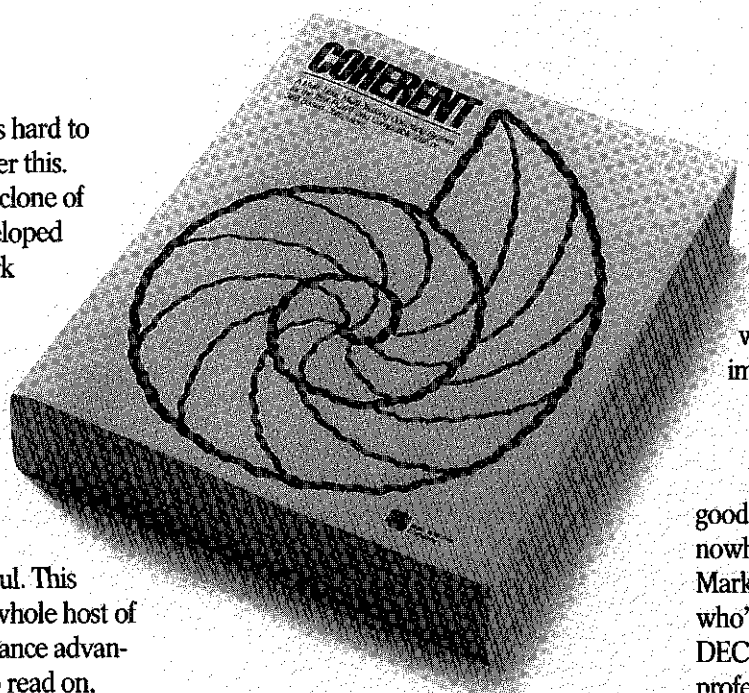
Goin Places Software
Suite 145
1202 N West 75th Street
Downers Grove IL 60516

For phone orders or info call: (708)759-1270

Circle 18 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Everything You Ever Wanted In UNIX. And Less. $99.95.*



OK. We know it's hard to believe. So just consider this. Coherent™ is a virtual clone of UNIX. But it was developed independently by Mark Williams Company. Which means we don't pay hundreds of dollars per copy in licensing fees.

What's more, Coherent embodies the original tenet of UNIX: small is beautiful. This simple fact leads to a whole host of both cost and performance advantages for Coherent. So read on, because there's a lot more to Coherent than its price.

SMALLER, FASTER...BETTER.

Everybody appreciates a good deal. But what is it that makes small so great?

For one thing, Coherent gives you UNIX capabilities on a machine you can actually afford. Requiring only 10 megabytes of disk space,

LESS IS MORE!	Coherent For the IBM-PC/AT and compatible 286 or 386 based machines.	Santa Cruz Operation's XENIX 286, Version 2.3.2
No. of Manuals	1	8
No. of Disks	4	21
Kernel Size	64K	198K
Install Time	20-30 min.	3-4 hours
Suggested Disk Space	10 meg	30 meg
Min. Memory Required	640K	1-2 meg
Performance*	38.7 sec	100.3 sec
Price	$99.95	$1495.00

*Byte Execl benchmark, 1000 iterations on 20 MHZ 386.
Hardware requirements: 1.2 meg 5¼" or 1.4 meg 3½" floppy, and hard disk. SCSI device driver available soon. Does not run on Microchannel machines.

Coherent can reside with DOS. So you can keep all your DOS applications and move up to Coherent. You can also have it running faster, learn it faster and get faster overall performance. All because Coherent is small. Sounds beautiful, doesn't it?

But small wouldn't be so great if it didn't do the job it was meant to do.

EVERYTHING UNIX WAS MEANT TO DO.

Like the original UNIX, Coherent is a powerful multi-user, multi-tasking development system. With a complete UNIX-compatible kernel which makes a vast world of UNIX software available including over a gigabyte of public domain software.

Coherent also comes with Lex and Yacc, a complete C compiler and a full set of nearly 200 UNIX commands including text processing, program development, administrative and maintenance commands.

And with UUCP, the UNIX to UNIX Communication Program that connects you to a world-wide network of free software, news and millions of users. All for the cost of a phone call.

We could go on, but stop we must to get in a few more very important points.

EXPERIENCE, SUPPORT AND GUARANTEES.

Wondering how something as good as Coherent could come from nowhere? Well it didn't. It came from Mark Williams Company, people who've developed C compilers for DEC, Intel, Wang and thousands of professional programmers.

We make all this experience available to users through complete technical support via telephone. And from the original system developers, too!

Yes, we know $99.95 may still be hard to believe. But we've made it fool-proof to find out for yourself. With a 60-day money-back no-hassles guarantee.

You have to be more than just a little curious about Coherent by now. So why not just do it? Pick up that phone and order today.

010-1-708-291-6700

ORDER BY FAX:

010-1-708-291-6750

Pay by Visa, MasterCard or Am Ex.
*Plus shipping to U.K.:
3-day DHL delivery, $31. U.S. Air Mail, $30.

60-DAY MONEY BACK GUARANTEE!

Mark Williams Company
60 Revere Drive
Northbrook, IL 60062, U.S.A.

Coherent is a trademark of Mark Williams Company. UNIX is a trademark of AT&T. XENIX is a trademark of Microsoft.

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Negotiating Tricks and Traps

By William H. Roetzheim

Believe it or not, you are involved in negotiations every day of your life. One obvious example is when you, as a computer consultant, discuss price with your client. Other less obvious examples occur in discussions about the terms of the contract, changes to software, and the establishment of teaming agreements. The best advice for any negotiator is to do your homework. In any negotiation, the person with the best preparation has a tremendous advantage.

The first steps in this process are to define and prioritize your objectives for the negotiation, and to attempt to do the same for your customer. Then gather and analyze facts that are relevant to your objectives. Having the facts at your fingertips allows you to use them to your advantage, and prevents the other side from using "facts" that you know to be false.

Finally, sit down and formulate a negotiating strategy, deciding which issues you will hold firm to and which you are willing to move on. Make your decisions about what you are willing to give up while you are under no pressure. When deciding what you will hold firm to, ask yourself if these issues are important enough to risk losing the contract. If the answer is no, then the item is negotiable.

NEGOTIATING STRATEGIES

Oddly enough, professional negotiators who work for large companies and government agencies are fond of using some of the same negotiating tricks that salesmen have used for ages. By educating yourself about these tricks, you will be able to spot them when they are used. Once a negotiating tactic has been recognized, it is easy to find a tactic that will counteract its effectiveness. The five techniques we will cover are:

1. The missing man
2. Lowballing
3. Divide-and-conquer
4. Silent intimidation
5. The opening gambit

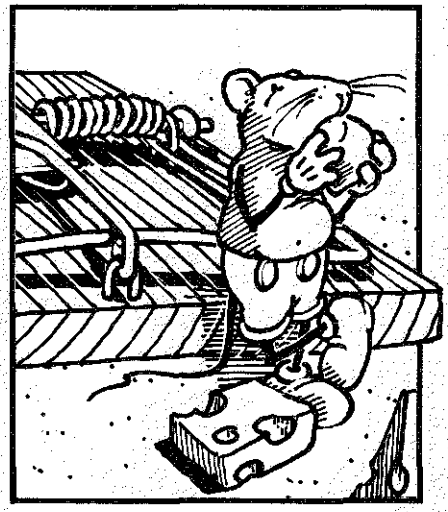


ILLUSTRATION BY JIM CARPENTER

THE MISSING MAN

The missing-man technique is a favorite of car salesmen the world over. My experience when I bought my last new car was a typical example:

A courteous but anxious salesman showed me the car and took me out for a test drive. I was escorted to a small, cramped room where some preliminary forms were filled out (to enhance my mental commitment to buying the car). Finally, the negotiations started in earnest. We discussed every item about the car, from the type of radio that was included to the thread pattern on the tires, before finally arriving at a mutually acceptable price two hours later. The salesman left the room to finish the paperwork, only to return five minutes later with a crestfallen expression on his face. The manager wouldn't approve the price, he stammered, looking at his shoe (this guy was *good*). The negotiations were then supposed to begin anew, starting at the price I had just agreed to!

As this story illustrates, the missing-man technique involves having someone else (anyone else) who must give final approval to any deal that is negotiated. In this way the hapless victim actually ends up negotiating his or her best deal, which then becomes the starting point for the next round of negotiations. The best counter to the missing man technique is to refuse to negotiate with anybody who doesn't have the final authority to make a commitment. If you are trapped by this technique, you can always hold fast to your original offer and refuse to budge, or you can appear to be very understanding and explain that, of course, any offer you negotiate is also subject to the approval of someone in *your* office. This is exactly the reason why most companies will not send their top person to conduct negotiations: they want to keep a built-in out for themselves.

LOWBALLING

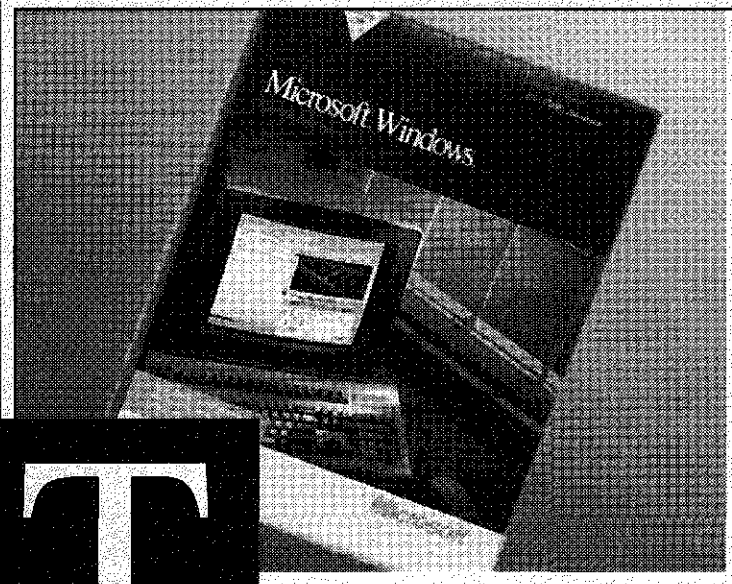
This is the technique most people think of when describing a tough negotiator. Lowballing simply involves making an extreme offer, then moving from this offer in very small increments. The negotiators may make some minor changes in your plan, then offer you half the money you originally quoted. After further discussion they go up, but only by a very small amount.

Lowballing is a risky strategy. It can result in a very good deal (pricewise) for the team using it or it can cause a negotiating deadlock in which no agreement is reached. It is also a difficult technique to defend against. My usual countertactic is to stand up, calmly explain that I am aware of the tactic they are using, and tell them that these negotiations are in serious danger of reaching a deadlock. I then ask them to reconsider their position and leave the room for a cup of coffee.

Quite often, 15 minutes to discuss the matter among themselves is enough to return them to their senses. An alternate approach is to change the basic premise of the negotiation. For example, you could stop discussing a

William H. Roetzheim is an MIS consultant with Booz, Allen, and Hamilton, and the author of several books. This article is based, with permission, on the author's book Proposal Writing for the Data Processing Consultant, *Prentice-Hall, 1986. Bill may be contacted at 13518 Jamul Drive, Jamul, CA 92035; (619) 669-6970.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



There is nothing in this industry that Windows 3.0 isn't going to change.

Bill Gates
PC Week, May 20, 1990

Microsoft introduced Windows™ 3.0 in May. Now we're selling 30,000 copies a week. In fact, this revolutionary product accounted for almost a third of our first quarter revenues.

Windows is impacting more than our bottom line. It's affecting corporate operating systems, LANs, servers and application programs around the world. And the hub of all this is Microsoft's Information Technology Integration Services (ITIS) operation. Want to help put better information at the fingertips of corporate users everywhere? Begin by exploring the information now at your fingertips.

Windows Consultants

We are looking for seasoned developers with experience in systems architecture, systems and applications design, and development experience in Windows. You should be a strong C programmer capable of both creating new applications and customizing and interfacing "shrink-wrapped" products in a multi-vendor environment.

To make it all connect, you'll need experience designing and implementing LAN systems, extensive knowledge of communications protocols and networking software (e.g. LAN Manager, SQL Server), and expertise solving micro-to-mainframe and network-to-network connectivity issues. Your 5+ years of experience, BS/MS in Computer Science, and strong knowledge of MS products will help you help our customers realize the full impact of Windows.

What do you say?

Everyone at Microsoft has a voice in the future of personal computing. Join our ITIS operation, and you'll find a lot of very important and attentive listeners.

Begin by sending your resume to: MICROSOFT CORPORATION, Recruiting, Dept. NHCON-191PJ, One Microsoft Way, Redmond, WA 98052-6399. No phone calls, please. We are an equal opportunity employer and are working toward a more culturally diverse workplace.

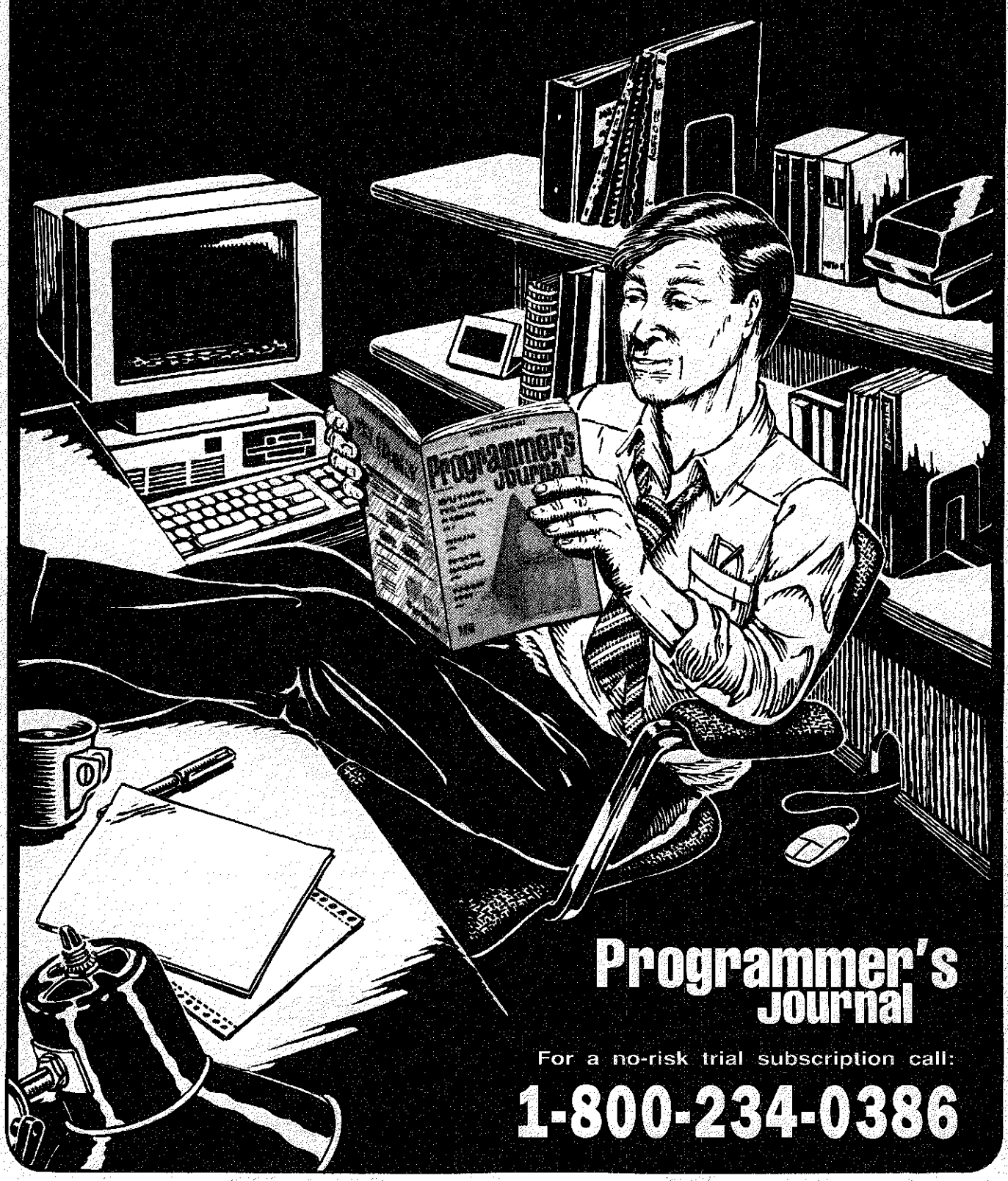
Microsoft®
Making it all make sense™

Trademarks are registered to their respective companies.

Circle 232 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

README.



Programmer's Journal

For a no-risk trial subscription call:

1-800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

fixed price for the work and begin talking about an hourly rate in its place.

DIVIDE AND CONQUER

This technique is simple. When facing a negotiating team, split them into two or more groups and get the groups working against each other. You don't even have to physically separate the people for this technique to work. Let me give you a quick example of how it's done:

I'm at one end of the table and your negotiating team is at the other. During the discussion, I begin to direct my attention to a member of your team I have identified as a weak link. I ask questions directly to this individual and attempt to get him or her to answer my questions. I continue this until the individual and I are conducting the negotiation, and you and the rest of your team are not even in the picture. I get a concession or two, then direct my attention to another individual identified as a weak link, ignoring both the first person and you. Soon I can have your entire team conducting individual negotiations with me and doing their utmost to get my attention (normally by throwing out some offer). I listen to these offers and quickly accept any and all that I feel are worthwhile. You thus give up the entire ship, one piece at a time.

The counter strategy to the divide-and-conquer ruse is simple, but it must be set up prior to the start of the negotiations and enforced strictly. Every negotiating team has one—and only one—mouth. No matter how many people are on your team, no one ever speaks except you. If a member of your team has something important to say, he or she passes you a note, or you call a five-minute break in order to talk in private.

SILENT INTIMIDATION

Silence makes people uncomfortable. They will do almost anything to end a silence, even if it means making outrageous deals. Let me give you a true example of this technique in action:

I'm negotiating a deal to buy a house that is for sale by the owners. We're all sitting around the dining-room table staring at my written offer, which is ridiculously low (lowballing). Nobody says anything for about 30 seconds, until the husband—I'll call him Mr. Smith—clears his throat and then says: "We've looked at your offer, and we think it's far too low, considering the appraised value of this house."

I don't say anything. Two or three minutes crawl by. Mrs. Smith says: "We could probably come down a little, but not nearly that much." I still don't say anything. Another minute passes before Mr. Smith can't stand it: "We might go as low as $85,000. That's $5,000 under our asking price and $10,000 below the appraised value."

Silence on my part. Minutes pass. They ask if they could be excused for a minute. A few minutes later they return

Tired of the CONFIG.SYS shuffle?

Let BOOTCON simplify your life.

"... *it's bulletproof ... The interface is well thought out. You need this program.*"
Dr. Jerry Pournelle, *Byte*, July 1990

"*Highly recommended.*"
John C. Dvorak, *PC Magazine*, July 1990

How many CONFIG.SYS and AUTOEXEC.BAT combinations do you have on your PC? BOOTCON eliminates the confusion by storing all your configuration information in just one set of configuration files. Each time you boot your PC you can select the configuration you want to use from a menu of up to 26 choices.

BOOTCON is ideal when you need to use conflicting programs such as Windows 3.0, QEMM/386, 386MAX, Soft-ICE, and Lotus 123 3.0. A default configuration will automatically be used if you do not make a selection within the specified time-out period. BOOTCON also provides an optional password protection feature.

BOOTCON lists for $59.95 and comes on both 5¼" and 3½" disks. It has an unconditional 30-day money-back guarantee. MS-DOS or PC-DOS 3.10 or later is required.

Modular Software Systems

115 West California Blvd., Suite 113 • Pasadena, CA 91105 (818) 794-7602

Windows 3.0 and MS-DOS are trademarks of Microsoft Corporation. PC-DOS is a trademark of International Business Machines Corporation. QEMM/386 is a trademark of Quarterdeck Office Systems. 386MAX is a trademark of Qualitas Corporation. Soft-ICE is a trademark of Nu-Mega Technologies. Lotus and 123 are trademarks of Lotus Development Corporation.

Circle 77 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and offer me the house for $79,900—more than $15,000 below the appraised value. This figure then becomes our starting point for active (and verbal) negotiations about terms, conditions, and so on. I didn't have to say a single word to find out their lowest price!

The silent-intimidation technique is extremely simple to counter, if you recognize it. You can defend either by using the same technique or by asking a series of simple, relatively trivial questions which force the person to enter into a discussion.

THE OPENING GAMBIT

This technique involves shocking your opponents so deeply that their negotiating strategies are forgotten in the ensuing panic. It goes something like this: The company representative enters the room and drops an "official" letter on the table which states that your prices are significantly higher than budgeted for and recommending that the procurement be canceled. You are left making rapid concessions to save the procurement from destruction when the letter may have absolutely no basis in fact!

When faced with an opening gambit where the entire rules of the game are changed in the 30 seconds before negotiations begin, I strongly recommend that you delay the negotiations to consider the implications of this change. You need time to discover how much truth there is in the gambit, and time to prepare a new strategy.

NEGOTIATING DOS AND DON'TS

In closing, keep the following tips in mind to help your negotiation experiences be profitable and enjoyable:

- Do promote an atmosphere of partnership and cooperation. Your objective is to arrive at an agreement which both you and the customer consider acceptable (win-win negotiating).
- Don't be a tough negotiator on insignificant points.
- Do use calm, collected body language which shows the other team that you are comfortable and in control.
- Don't try to prove that you are 100% correct, even if you are. Your objective is to arrive at an agreement, and this objective is hurt, not helped, by making your opponent lose face.
- Do observe the other team with your eyes, ears, and intuition. Negotiators will often communicate more to you through subtle intonation and body language than through words.
- Don't look tired, even if you are.
- Do make good eye contact throughout the negotiations. It gives the other team the impression that you are being open and honest with them.
- Don't send your company's senior executive to the negotiations, or else make it clear that a "partner" must approve all deals. This opens the door for the missing-man technique.
- Do have only one spokesperson for your team. PJ

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3
Excel • Paradox • Informix • Q&A • Oracle • Xdb
askSam • SC4 • Clarion • DIF • SYLK • Symphony
ACT! • Platinum • DAC • and other application files
IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries
Btrieve • c-tree • EBCDIC • Packed data
. . . and more!

- *No programming-completely menu driven!*
- *Select and rearrange records, fields and characters!*
- *Instant script file creation for total batch processing!*
- *Map new data structures automatically by name!*
- *Ideal for OEM applications!*

". . . an invaluable integration tool." -Byte
". . . more than worth its price." -PC Magazine
". . . a surefire winner." -Computer Language

Standard $99 - dBASE, ASCII, 1-2-3, Q&A. Enable, DIF, askSam & merge files for WordPerfect, Word & WordStar
Professional $199 - Standard formats plus Binary/EBCDIC, SC4, SYLK Paradox, R:BASE, Clarion, Excel, c-tree & Btreve
Advanced $299 - Standard & Professional formats plus Informix, Xdb, Oracle & C-ISAM

Call 800/444-1945 or 512/482-0824
MC/VISA • AMEX • COD • Corp. PO.
available for LANs, Xenix & Unix
Tools & Techniques Inc. • 1620 W 12 • Austin TX 78703

T&T TOOLS & TECHNIQUES

Circle 327 on Reader Service Card

Tired of the limitations of MS-DOS?
Thinking about switching to another OS?
Well DON'T, now there is:

PRO-CLI v4.1

CLI is a tiny, intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS & UNIX, to MS-DOS without the tremendous costs and additional training.

CLI-Types: ALIASs, LOGICALs, SYMBOLs, LEXICAL-FUNCTIONs and STREAMs are available to the command line, batch programs, & all existing application programs. CLI is 100% MS-DOS compatible, and NO changes are required in any software to reference any CLI-Types.

Some of the features offered by PRO-CLI include:

- batch programs with numeric & string processing
- facilities for source code and project management
- transparent task-to-task communications
- file & device management from the command line
- command line editing, history, and recall
- flexible system installations & re-configurations
- user configurable PRO-CLI resources

For the first time, designers, developers, and managers can coordinate software development & production efforts without the major restrictions of MS-DOS.

Call or Write for Additional Information!

Trademarks: VMS of D.E.C. MS-DOS of Microsoft UNIX of AT&T

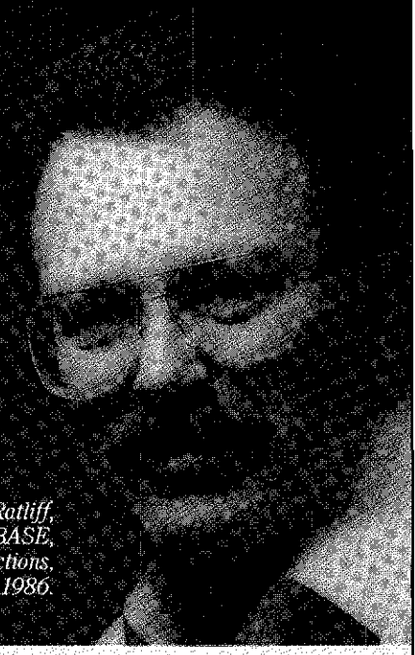
Alphax Systems, Inc.
1005 Sussex Blvd.
Broomall, PA 19008
(215) 543-2658

Price: $ 149
S/H: $ 6
Major Credit Cards Accepted

Circle 345 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.



Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.

You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages. Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.

New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

BRIEF 3.0

The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $249 plus $6 shipping (U.S. and Canada). 30-day money-back guarantee.

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____
Paying by ☐ Check ☐ MC ☐ VISA
Card # _____ Exp. date _____
Signature _____
Disk size ☐ 5-1/4" ☐ 3-1/2" Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492

Extension 496

Massachusetts & outside U.S. call 617-337-6963

Circle 47 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VGA color cycling

The intricacies of color cycling, paging, and loading the DAC

By Michael Abrash

Installment 22: *In which, with a sense of bemusement, we delve into the intricacies of loading the DAC.*

Sometimes the harder you try, the less you accomplish. Brute force is fine when it suffices, but when it does not, finesse and alternative approaches are called for. Such is the case with rapidly cycling through colors by repeatedly loading the VGA's Digital to Analog Converter (DAC). No matter how much you optimize your code, you just can't reliably load the whole DAC cleanly in a single frame, so it's best to find other ways to use the DAC to cycle colors. What's more, BIOS support for DAC loading is so inconsistent that it's unusable for color cycling: direct loading through the I/O ports is the only way to go. We'll see why as we explore color cycling, and then we'll clean up some odds and ends about VGA color.

COLOR CYCLING

As we've learned in past installments, the VGA's DAC contains 256 storage locations, each holding one 18-bit value representing an RGB color triplet set up as 6 bits per primary color. Each pixel generated by the VGA is fed into the DAC as an 8-bit value (refer to the last two installments to see how pixels become 8-bit values in non-256-color modes) and each 8-bit value is used to look up one of the 256 values stored in the DAC. The looked-up value is then converted to analog red, green, and blue signals and sent to the monitor to form one pixel.

That's straightforward, and we've produced some pretty impressive color effects by loading the DAC once and then playing with the 8-bit path into the DAC. Now we want to generate color effects by dynamically changing the values stored in the DAC in real time, a technique I'll call color cycling. The potential of color cycling should be obvious: Smooth motion can easily be simulated by altering the colors in an appropriate pattern, and all sorts of changing color effects can be produced without modifying a single bit of display memory. For example, a sunset can be made to color and darken by altering the DAC locations containing the colors used to draw the sunset, or a river can be made to appear to flow by cycling through the colors used to draw it. This leaves only one question: how do we load the DAC smoothly in real time?

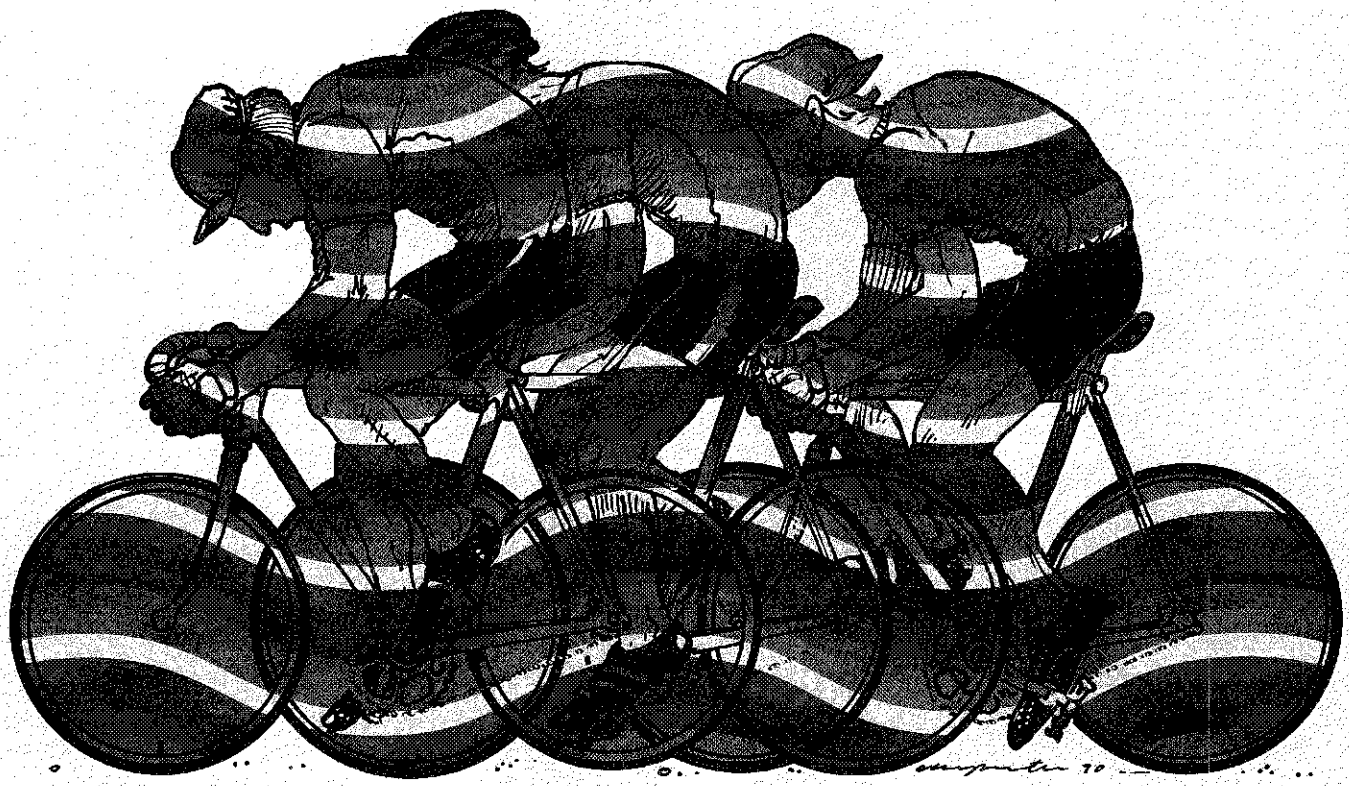
As far as I know, you can't load the *entire* DAC—all 256 locations—frame after frame without producing distressing on-screen effects on at least some computers. In non-256-color modes, loading the DAC quickly enough to cycle all displayed colors (of which there are 16 or fewer) is indeed possible, so color cycling can be used successfully to cycle all colors in these modes. On the other hand, color paging (which flips among a number of color sets stored within the DAC, as discussed in the last installment) can be used in non-256-color modes to produce many of the same effects as color cycling. Color paging is considerably simpler and typically more reliable then color cycling, so color paging is generally superior whenever it's available. In short, color cycling is really the method of choice for dynamic color effects only in 256-color mode—but, regrettably, color cycling is at its least reliable and capable in that mode, as we'll see next.

THE PROBLEM

The problem with loading the entire DAC repeatedly is that it contains 256 color storage locations, each loaded via either 3 or 4 **outs** (more on which below). This means that at least 768 **outs** are needed to load the entire DAC. That many **out** instructions take up a lot of time, all the more so because the DAC is usually on the expansion bus, where wait states are inserted in fast computers. In an 8MHz AT, the 768 **outs** alone would take 288 microseconds, and the necessary data

Contributing Editor Michael Abrash works for Metagraphics of Scotts Valley, California. He is the author of Power Graphics Programming *and* Zen of Assembly Language. *Contact Michael at 7 Adirondack St., S. Burlington, VT 05403, or via MCI Mail, user name* MABRASH.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



loading and looping would take in the ballpark of 1800 microseconds more, for a minimum of 2 milliseconds total.

DON'T LET IT SNOW

As it so happens, the DAC should be loaded only during vertical non-display time: that is, the time between the end of displaying the bottom border and the start of displaying the top border, when no video information at all is being sent to the screen by the DAC. Otherwise, small dots of snow appear on the screen. While an occasional dot wouldn't be a problem, the constant DAC loading required by color cycling would produce a veritable snowstorm on the screen. By the way, I do mean "border," not "bitmap"; the overscan pixels pass through the DAC just like the pixels controlled by the bitmap, so you can't even load the DAC while the border color is being displayed without getting snow.

The start of vertical non-display time itself is not easy to find, but the leading edge of the vertical sync pulse is readily detected via bit 3 of the Input Status 1 register at 3DAh. When bit 3 is 1, the vertical sync pulse is active. Conveniently, the vertical sync pulse starts part of the way through but not too far into vertical non-display time, so it serves as a handy way to tell when it's safe to load the DAC without producing garbage on the screen.

So we wait for the start of the vertical sync pulse, then begin to load the DAC. There's a catch, though. On many computers—386s under some circumstances, 286s under most circumstances, and 8088s under all circumstances—there just isn't enough time between the start of the vertical sync pulse and the end of the vertical non-display time to perform that two-millisecond-plus load of all 256 DAC locations. That's the crux of the problem with the DAC, and soon we'll get to a tool that will let you explore for yourself the extent of the problem on computers you're interested in. First we must address another DAC loading problem, though: the BIOS.

LOADING VIA THE BIOS

The DAC can be loaded either directly or through subfunctions 10h (for a single DAC register) or 12h (for a block of DAC registers) of the BIOS video service interrupt 10h, function 10h. For cycling the contents of the entire DAC, the block-load function (invoked by executing INT 10h with AH = 10h and AL = 12h to load a block of CX DAC locations, starting at location BX, from the block of RGB triplets—three bytes per triplet—starting at ES:DX) would be the better of the two, due to the greater efficiency of calling the BIOS once rather than 256 times. At any rate, we'd like to use one or the other of the BIOS functions for color cycling, because we know that using a BIOS function rather than accessing hardware directly helps in avoiding compatibility problems. In the case of color cycling, however, it is emphatically **not** possible to use either of the BIOS functions, for they have problems. Serious problems.

The difficulty is this: IBM's BIOS specification describes exactly how the parameters passed to the BIOS control the loading of DAC locations, and all clone BIOSes meet that specification scrupulously. This is to say that if you invoke INT 10h, function 10h, subfunction 12h with a given set of parameters, you can be sure you will end up with the same values loaded into the same DAC locations on all VGAs from all vendors. IBM's spec does not, however, describe whether to wait for vertical retrace before loading the DAC, nor does it mention whether to leave the video enabled while loading the DAC. These omissions have left cloners free to opt for whatever approach they desire—and, alas, every cloner seems to have selected a different approach.

I tested four clone VGAs from different manufacturers, some in a 20MHz 386 machine and some in a 10MHz 286 machine. Two of the four waited for vertical retrace before loading the DAC and two didn't. Two of the four blanked the display while loading the DAC, resulting in flickering bars across the screen. One of them showed speckled pixels spattered across the top of the screen while the DAC was being loaded. Not one was able to load all 256 DAC locations without showing *some*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CODE RUNNER

Create ultra compact and fast TSRs with full DOS access!

The most powerful tool available for creating feature packed, tiny TSRs in C or assembler. Works with C compilers from Microsoft, Borland, Watcom and Zortec.

CodeRunneR Features

- Auto recovery of Initialization Code and Data
- All functions hand crafted highly optimized assembly language routines
- Function Level Granular
- Create MultiTasking programs
- Scheduler supports > 10,000 events
- Assess TSRs via up to 256, dynamic hot-keys or interrupts
- BCD Floating Point Library
- Source for TSRs: Help, Programmers Calculator, Business Calculator, ASCII/Color, Date/Alarm & Trap/Debug

CodeRunneR $149.00

Call our BBS for FREE sample TSRs, benchmarks & support. See for yourself how powerful and tiny CodeRunneR TSRs can be.
BBS: (508) 875-8009

NEW

PDK#3

- Spawn (EXEC) large programs from a hosting TSR (<1.5 KBytes)
- Page foreground application or hosting TSR to LIM* or Disk
- Pop-up TSRs over graphics screens!
- Save & Restore COM ports, Mouse and Math Co-Processor states.
- Turn ANY existing program into a TSR (without source code -- even Lotus 1-2-3)! * Needs PDK#1

PDK#1

- COM: Ultra fast interrupt driven multichannel COM library with background file transfers
- LIM: Transparently move TSR code &/or data to LIM
- Flexible printer SPOOLer
- Program registration and protection functions

PDK#1&3 $99.00 each

NO ROYALTIES. MONEY BACK offer and source code available.
Order today by Phone, FAX or BBS.
MasterCard, VISA & AMEX accepted.

Microsystems Software, Inc.™
600 Worcester Rd. Framingham, MA 01701
(508) 626-8511 FAX: (508) 626-8515

CodeRunneR is a Registered Trademark of Microsystems Software, 1-2-3 is a Trademark of Lotus.

Circle 40 on Reader Service Card

sort of garbage on the screen for at least one frame, but that's not the BIOS's fault; it's a problem endemic to the VGA.

The above findings lead me inexorably to the conclusion that the BIOS should not be used to load the DAC dynamically. If you're loading the DAC just once in preparation for a graphics session—sort of a DAC mode set—by all means load by way of the BIOS. No one will care that some garbage is displayed for a single frame—I have boards that bounce and flicker and show garbage every time I do a mode set, and the garbage produced by loading the DAC once is far less noticeable. If, however, you plan to load the DAC repeatedly for color cycling, avoid the BIOS DAC load functions like the plague. They will bring you nothing but trouble.

As but one example of the unsuitability of the BIOS DAC-loading functions for color cycling, imagine that you want to cycle all 256 colors 70 times a second, which is once per frame. You would normally wait for the start of the vertical sync signal (marking the end of the frame), then call the BIOS to load the DAC. On some boards—boards with BIOSes that don't wait for vertical sync before loading the DAC—that will work pretty well; you will, in fact, load the DAC once per frame. On other boards, however, it will work very poorly indeed; your program will wait for the start of vertical sync, and then the BIOS will wait for the start of the *next* vertical sync, so the DAC gets loaded only once every *two* frames.

Sadly, there's no way for you to know which sort of BIOS is installed in a particular computer, so unless you can always control the brand of VGA your software will run on, you can't afford to color cycle by calling the BIOS. This is not to say that loading the DAC directly is a picnic either, as we'll see next.

DIRECT LOADING

So we must load the DAC directly in order to perform color cycling. This is accomplished by **out**ing the number of the DAC location to be loaded to the DAC Write Index register at 3C8h and then performing three **out**s to write an RGB triplet to the DAC Data register at 3C9h. This approach must be repeated 256 times to load the entire DAC, requiring 1K **out**s in all.

There is another, faster approach, but one that has risks. After an RGB triplet is written to the DAC Data register, the DAC Write Index register automatically increments to point to the next DAC location, and this repeats indefinitely as successive RGB triplets are written to the DAC. The entire DAC can be loaded with just 769 **out**s: one **out** to the DAC Write Index register and 768 **out**s to the DAC Data register.

So what's the drawback? Well, imagine that as you're loading the DAC, an interrupt-driven TSR (such as a program switcher or multitasker) activates and writes to the DAC. You could end up with quite a mess on the screen, especially when your program resumes and continues writing to the DAC—probably to the wrong locations. No problem, you say; just disable interrupts for the duration. Good idea—but it takes much longer to load the DAC than interrupts should be disabled for. If, on the other hand, you set the index for each DAC location separately, you can disable interrupts 256 times, once as each DAC location is loaded, without problems.

As I commented two installments back, I don't have any gruesome tale to relate that mandates taking the slower but safer road and setting the index for each DAC location separately while interrupts are disabled. I'm just hypothesizing as to what ghastly mishaps *could* happen. However, it's been my experience that anything that can happen on the PC *does* happen eventually; there are just too dang many PCs out there for it to be otherwise. However, load the DAC any way you like; just don't blame me if you get a call from someone who claims that your program sometimes turns their screen into something resembling month-old yogurt. It's not really your fault, of course—but try explaining that to them!

A TEST PROGRAM

The choice of how to load the DAC is yours. Given that I'm not providing you with hard-and-fast rules (mainly because there don't seem to be any), what you need is a tool so that you can experiment with various DAC-loading approaches for yourself. That tool is what you'll find in Listing 1.

Listing 1 draws a band of vertical lines across the screen, each one pixel

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

wide. The attribute of each vertical line is one greater than that of the preceding line, so there's a smooth gradient of attributes from left to right. Once everything is set, the program begins to cycle the colors stored in the number of DAC locations specified by the *CYCLE_SIZE* equate; as many as all of the 256 DAC locations can be cycled. (Actually, *CYCLE_SIZE*-1 locations are cycled, because location 0 is kept constant to keep the background and border colors from changing, but *CYCLE_SIZE* locations are loaded. It's the number of locations we can load without problems that we're interested in.)

The big question is, how does Listing 1 cycle colors? Via the BIOS? Directly? With interrupts enabled or disabled? The answer is, however you like. Four equates at the top of Listing 1 select the sort of color cycling to be performed; by changing these equates and *CYCLE_SIZE*, you can get a feel for how well different approaches to color cycling work with the combination of computer system and VGA you care to test.

The *USE_BIOS* equate is fairly simple. Set *USE_BIOS* to 1 to load the DAC via the block-load-DAC BIOS function, or to 0 to load the DAC directly with **outs**.

If *USE_BIOS* is 1 and *WAIT_VSYNCH* is 1, the program waits for the leading edge of vertical sync before loading the DAC. If *WAIT_VSYNCH* is 0, the program doesn't wait before loading. The effect of setting or not setting *WAIT_VSYNCH* depends on whether the BIOS of the VGA waits for vertical sync before loading the DAC. You may end up with a double wait, causing color cycling to proceed at half speed; you may end up with no wait, causing cycling to occur far too rapidly (and almost certainly with hideous on-screen effects); or you may actually end up cycling at the proper one-cycle-per-frame rate.

If *USE_BIOS* is 0, *WAIT_VSYNCH* still applies. However, you will always want to set *WAIT_VSYNCH* to 1 when *USE_BIOS* is 0; otherwise, cycling will occur much too fast, and a good deal of continuous on-screen garbage is likely to make itself evident as the program loads the DAC non-stop.

SLOW BUT SAFE

GUARD_AGAINST_INTS comes into play when *USE_BIOS* is 0 . If *GUARD_AGAINST_INTS* is 1, each DAC location is loaded as follows: interrupts are disabled, the DAC Write Index register is set appropriately, the RGB triplet for the location is written to the DAC Data register, and interrupts are enabled. This is the slow but safe approach. Matters get even more interesting if *GUARD_AGAINST_INTS* is 0.

In that case, and if *NOT_8088* is 0, then an autoincrementing load is performed in a straightforward fashion: the DAC Write Index register is set to the index of the first location to load and the RGB triplet is sent to the DAC via three **lodsb/out** DX,AL pairs, with **loop**

"Writing a TSR is exceptionally easy" ... and now it's inexpensive too!

$49

Now you can turn Turbo Pascal programs into rock solid TSRs with ease. **TSRs Made Easy** lets you create conventional TSRs or swapping TSRs that use only 6K of RAM. **TSRs Made Easy** provides ■ TSR swapping to EMS, XMS, or disk ■ selectable hot keys ■ keyboard macros ■ unloadable TSRs ■ 8087 TSR support ■ interface to transient programs ■ ISR handling, and more.

TSRs Made Easy includes full source, complete documentation, and plenty of small example and demo programs. You pay no royalties.

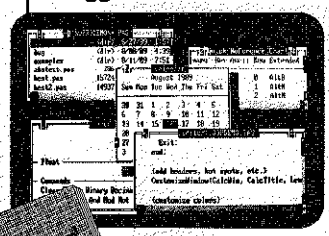
"Writing a TSR. . .is exceptionally easy. The documentation is extremely readable and well done."
Computer Language, May 1990

One of programming's most formidable tasks is now very simple. . . and very affordable!

TSRs Made Easy, only $49.

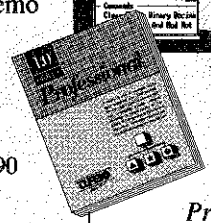
TSRs Made Easy has exactly the same TSR routines as OPro. TSRs Made Easy requires Turbo Pascal 6.0, 5.5 or 5.0. OPro requires Turbo 6.0 or 5.5. 5.25" and 3.5" disks included. Add $5 per order for standard shipping in U.S./Canada. Call for other shipping charges. Registered owners of OPro may update to version 1.1 for $20, include your serial number.

New! Object Professional for Turbo Pascal 6.0



Object Professional version 1.1 is fully updated for Turbo Pascal 6.0. New are SAA/CUA style dialog boxes, draggable windows, XMS/EMS 4 support, and more.

Object Professional includes over 100 object types that will multiply your productivity. Included are windowing and menu systems, menu and data entry screen generators, data object types, and routines that provide swapping TSRs.



You'll get up to speed fast with clear documentation, on-line help, full source code, and hot demo programs.

"The range of objects is fantastic. Object Professional could save you man-years of effort."
Jeff Duntemann

Object Professional, only $189.

Call toll-free to order: 1-800-333-4160

TURBO Power

9AM–5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608, fax to (408) 438-8610, or send mail to CompuServe ID 76004,2611
TurboPower Software PO Box 66747 Scotts Valley, CA 95067-0747

Circle 25 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

repeating the process for each of the locations in turn.

If *NOT_8088* is 1, indicating that the processor is a 286 or better, then after the initial DAC Write Index value is set, all 768 DAC locations are loaded with a single **rep outsb**. This is clearly the fastest approach, but has the obvious drawback of not working in millions of computers—and the ones that it doesn't work in are precisely those computers in which speed is most critical.

My experience with Listing 1 indicates that it is not possible to load all 256 locations cleanly if interrupts are guarded against and autoincrementing is not used, even on a 20MHz 386. Because **outs** are notoriously slow on a

LISTING 1

```
; Fills a band across screen with vertical bars in all 256 attributes
; then cycles a portion of the palette until a key is pressed.
; Assemble with MASM 5.1 or TASM 2.0.
USE_BIOS equ    1       ;set to 1 to use BIOS functions to access the
                        ; DAC, 0 to read and write the DAC directly
GUARD_AGAINST_INTS equ 1 ;1 to turn off interrupts and set write index
                        ; before loading each DAC location, 0 to rely
                        ; on the DAC auto-incrementing
WAIT_VSYNC equ  1       ;set to 1 to wait for the leading edge of
                        ; vertical sync before accessing the DAC, 0
                        ; not to wait
NOT_8088 equ    0       ;set to 1 to use REP INSB and REP OUTSB when
                        ; accessing the  DAC directly, 0 to use
                        ; IN/STOSB and LODSB/OUT
CYCLE_SIZE equ  256     ;# of DAC locations to cycle, 256 max
SCREEN_SEGMENT equ 0a000h ;mode 13h display memory segment
SCREEN_WIDTH_IN_BYTES equ 320 ;# of bytes across the screen in mode 13h
INPUT_STATUS_1  equ 03dah ;input status 1 register port
DAC_READ_INDEX  equ 03c7h ;DAC Read Index register
DAC_WRITE_INDEX equ 03c8h ;DAC Write Index register
DAC_DATA equ    03c9h     ;DAC Data register
if NOT_8088
        .286
endif   ;NOT_8088
        .model  small
        .stack  100h
        .data
;Storage for all 256 DAC locations, organized as one three-byte
; (actually three 6-bit values; upper two bits of each byte aren't
; significant) RGB triplet per color.
PaletteTemp     db      256*3 dup(?)
        .code
start:  mov     ax,@data
        mov     ds,ax
;Select VGA's standard 256-color graphics mode, mode 13h.
        mov     ax,0013h        ;AH = 0: set mode function,
        int     10h             ; AL = 13h: mode # to set
;Read all 256 DAC locations into PaletteTemp (3 6-bit values, one
; each for red, green, and blue, per DAC location).
if WAIT_VSYNC
;Wait for the leading edge of the vertical sync pulse; this ensures
; that we read the DAC starting during the vertical non-display
; period.
        mov     dx,INPUT_STATUS_1
WaitNotVSync:                   ;wait to be out of vertical sync
        in      al,dx
        and     al,08h
        jnz     WaitNotVSync
WaitVSync:                      ;wait until vertical sync begins
        in      al,dx
        and     al,08h
        jz      WaitVSync
endif   ;WAIT_VSYNC
if USE_BIOS
        mov     ax,1017h        ;AH = 10h: set DAC function,
                                ; AL = 17h: read DAC block subfunction
        sub     bx,bx           ;start with DAC location 0
        mov     cx,256          ;read out all 256 locations
        mov     dx,seg PaletteTemp
        mov     es,dx
        mov     dx,offset PaletteTemp ;point ES:DX to array in which
                                ; the DAC values are to be stored
        int     10h             ;read the DAC
else    ;!USE_BIOS
 if GUARD_AGAINST_INTS
        mov     cx,CYCLE_SIZE   ;# of DAC locations to load
        mov     di,seg PaletteTemp
        mov     es,di
        mov     di,offset PaletteTemp ;dump the DAC into this array
        sub     ah,ah           ;start with DAC location 0
DACStoreLoop:
        mov     dx,DAC_READ_INDEX
        mov     al,ah
        cli
        out     dx,al           ;set the DAC location #
        mov     dx,DAC_DATA
        in      al,dx           ;get the red component
        stosb
        in      al,dx           ;get the green component
        stosb
        in      al,dx           ;get the blue component
        stosb
        sti
        inc     ah
        loop    DACStoreLoop
 else   ;!GUARD_AGAINST_INTS
        mov     dx,DAC_READ_INDEX
        sub     al,al
        out     dx,al           ;set the initial DAC location to 0
        mov     di,seg PaletteTemp
        mov     es,di
        mov     di,offset PaletteTemp ;dump the DAC into this array
        mov     dx,DAC_DATA
  if NOT_8088
        mov     cx,CYCLE_SIZE*3
        rep     insb            ;read CYCLE_SIZE DAC locations at once
  else  ;!NOT_8088
        mov     cx,CYCLE_SIZE   ;# of DAC locations to load
DACStoreLoop:
        in      al,dx           ;get the red component
        stosb
        in      al,dx           ;get the green component
        stosb
        in      al,dx           ;get the blue component
        stosb
        loop    DACStoreLoop
  endif ;NOT_8088
 endif  ;GUARD_AGAINST_INTS
endif   ;USE_BIOS
;Draw a series of 1-pixel-wide vertical bars across the screen in
; attributes 1 through 255.
        mov     ax,SCREEN_SEGMENT
        mov     es,ax
        mov     di,50*SCREEN_WIDTH_IN_BYTES ;point ES:DI to the start
                                        ; of line 50 on the screen
        cld
        mov     dx,100          ;draw 100 lines high
RowLoop:
        mov     al,1            ;start each line with attr 1
        mov     cx,SCREEN_WIDTH_IN_BYTES ;do a full line across
ColumnLoop:
        stosb                   ;draw a pixel
        add     al,1            ;increment the attribute
        adc     al,0            ;if the attribute just turned over to 0
                                ; increment it to 1 because we're not going
                                ; to cycle DAC location 0, so
                                ; attribute 0 won't change
        loop    ColumnLoop
        dec     dx
        jnz     RowLoop
;Cycle the specified range of DAC locations until a key is pressed.
CycleLoop:
;Rotate colors 1-255 one position in the PaletteTemp array;
; location 0 is always left unchanged so that the background
; and border don't change.
        push    word ptr PaletteTemp+(1*3)      ;set aside PaletteTemp
        push    word ptr PaletteTemp+(1*3)+2    ; setting for attr 1
        mov     cx,254
        mov     si,offset PaletteTemp+(2*3)
        mov     di,offset PaletteTemp+(1*3)
        mov     ax,ds
        mov     es,ax
        mov     cx,254*3/2
        rep     movsw                   ;rotate PaletteTemp settings
                                        ; for attrs 2 through 255 to
                                        ; attrs 1 through 254
        pop     bx                      ;get back original settings
        pop     ax                      ; for attribute 1 and move
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Witness Software Development History.

In 1911, the greatest scientists in the world came to Brussels for The First Solvay Congress. It was the first time that Einstein, Curie, Planck, and Poincaré could be heard in one place at one time. When it ended, a new era in modern science began.

In 1991, the most prominent software developers in the world will descend upon Silicon Valley for Software Development '91. They will reveal new ways to achieve dramatic gains in software development productivity. When it's over, the next era of application development will begin.

The best and brightest programmers in the country want to show you how to write cleaner, faster, more elegant code—at Software Development '91, February 12-15, 1991 at the Santa Clara Convention Center. In just three-and-a-half days, they will transform the way you work by revealing new tricks, techniques, and technologies you can use as soon as you get back to work. You'll put more fun into your programming while you improve application quality.

Why You Should Attend

You re-energize your programming activity, because you learn ways to eliminate mundane tasks and increase your creative power. This is the most efficient way to stay on top of the sweeping changes taking place in software development. No other single event gives you access to the advice and counsel of so many industry leaders.

A partial list of scheduled speakers:

Jon Bentley • Grady Booch • Jon Carolan • Larry Constantine • Brad Cox • Bruce Eckel • Dick Gabriel • Jon Hopkins • Stan Kelly-Bootle • Steve Mellor • Tom Plum • William Roetzheim • Dave Schmitt • Zack Urlocker • Steve Weiss

Tour the Exhibits to Witness Hot New Products in Action

With almost 200 exhibits you'll be able to try out and compare all the newest products designed to help you develop your applications. Put products to the test with hands-on demonstrations. Talk to knowledgeable vendors from companies like Ashton-Tate, Borland, Digitalk, Greenleaf Software, Intel, Jensen & Partners, Microsoft, Novell, ParcPlace Systems, Quarterdeck, Watcom, and Zortec to find out how products are changing, what's hot, and what to look for in the months ahead.

Authoritative Sponsorship

Software Development '91 is sponsored by Miller Freeman Publications, publishers of *Computer Language, UNIX Review, AI Expert, Database Programming and Design, Embedded Systems Programming, LAN Magazine,* and *Software Development International.* You'll find the same high quality you've come to expect from these magazines at Software Development '91.

SOFTWARE DEVELOPMENT '91

Santa Clara Convention Center
February 12-15

Send Today For More Information

Call: (415)995-2472
FAX: (415)267-7683
Or write to:
Software Development Seminars,
500 Howard Street,
San Francisco, CA 94105

Circle 167 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386, I ran the code on a 10MHz 286 and found that even autoincrementing doesn't do the trick there. On a 10MHz 286, only the **rep outsb** approach allows all 256 DAC locations to be loaded without garbage. (Unrolling the loop could improve the looping approaches considerably.) I didn't have a PC or 6MHz AT handy, but I shudder to think how dismal the results might have been.

GOING FOR SPEED

I'm not going to report how many DAC locations I could successfully load with each of the various approaches because I don't have enough data points to make reliable suggestions, and I don't want you acting on my comments and running into trouble down the pike.

You now have a versatile tool with which to probe the limitations of various DAC-loading approaches. Use it to perform your own tests on a sampling of the slowest hardware configurations that you expect your programs to run on, then leave a generous safety margin. One thing is certain, though—unless your code runs only on very fast computers or you choose not to guard against interrupts, you're not going to be able to cycle all 256 DAC locations cleanly once per frame. If you have to support 8088 machines, you won't even come close, no matter how good your code is. That's why I said at the outset that brute force isn't appropriate to the task of color cycling. That doesn't mean that color cycling can't be used, just that subtler approaches must be employed. Let's look at some of those alternatives.

APPROACHES THAT WORK

First of all, I'd like to point out that when color cycling does work, it's a thing of beauty. Assemble Listing 1 so that it doesn't use the BIOS to load the DAC, doesn't guard against interrupts, and uses 286-specific instructions if your computer supports them. Then tinker with *CYCLE_SIZE* until the color cycling is perfectly clean on your computer. Color cycling looks stunningly smooth, doesn't it? And this is crude color cycling, working with the default color set; switch over to a color set that gradually works its way through various hues and saturations, and you could get something that looks for all the world like true-color animation (albeit working with a small subset of the full spectrum at any one time).

So how can we take advantage of color cycling within the limitations of loading the DAC? The simplest approach, and my personal favorite, is to cycle a portion of the DAC while using the rest of the DAC locations for non-cycling purposes. For example, you might allocate 32 DAC locations to the aforementioned sunset, reserve 160 additional locations for use in drawing a static mountain scene, and employ the remaining 64 locations to draw images of planes, cars, and the like in the foreground. The 32 sunset colors could be

WINDOWS APPLICATIONS IN 1/10th TIME!

* YOU CAN RAPIDLY PROTOTYPE FOR WINDOWS 3.0!
* Generates C Code for Complete SAA CUA User Interfaces.
* Does DATA VALIDATIONS and Spreadsheet-like TABLES.
* Ports in Minutes to DOS,Windows 3.0 and OS/2.
* Easy to Learn. Be MORE Productive in a few hours.
* FRONT ENDS to 3270/Other Hosts,LAN,SQL Server,Sybase, Informix,Oracle,BTrieve,dBVista,CIndex,Paradox,dBase....

Instant Prototyper. Both DOS & Windows,only $99.
PROTOTYPES:fancy,data oriented,50+Windows in one week.
ANIMATES:Windows/dialogs,Menus,Fields,Tables,Text File Editor,Help,Icons,radio/check/push buttons,pick/lists/combo boxes,Scroll bars,Bitmaps.VALIDATES:Custom/cross/lookup, display,required,picture(A9!LXB#Z$),range,date,time,money...

Instant Windows. Windows 3.0 $495.DOS $249.
Includes Prototyper,50+Function Library,350p Manual,C Code Generator (just add your DBMS/LAN/Internals) and examples.

Instant Windows Combo Pack. $995.
Includes Prototyper,Instant Windows for DOS & Windows,DLL, Linker,Dynamic Forms,250+Functions,1 Yr.Maintenance/Gold Level Tech.Support,4 hrs Premium Consulting,600p Manual.

CALL 415-324-9552 NOW! No Royalties.
No Runtimes.QUALITY Product,Tech.Support and Consulting.
WinSoft. 1016 E.El Camino Real#216,Sunnyvale CA 94087.

Circle 55 on Reader Service Card

Break Free !!!

Before

Save your Time, Money, and Sanity with our tool series for Windows 3.0 and OS/2.

	DLL	Source
SoftFIELDS™ Robust replacement Dialog Controls including: date, time, phone, money ssn, virtual multicolumn listbox, button, icon, scroll bar, user defined.	149	449
SoftDDE™ Effortlessly add DDE support to your application. A clean and elegant interface with Excel, Wingz, Toolbook, Ami, and other major Windows Apps.	149	449
SoftPAGETOOL™ The foundation of every interactive graphical design application. Create Page editors similar to CorelDRAW and Toolbook. All Required Toolboxes are included.	249	749
Soft2GUI™ Complete character mode GUI for the second (mono) monitor under Windows 3.0. Redirect input between monitors. A great way to display real time data.	149	449
SoftDPMI™ No need for assembly. Complete DPMI API in a callable DLL. Now you can access DPMI from Wingz or Toolbook scripts.	99	199

Call TODAY for complete list of Value Added Software Modules
Supports MSC, Turbo C++, ToolBook and Windows SDK Dialog Editor

SoftWorks International
P. O. Box 8628
Atlanta, Ga 30306
To Order Call:
(404) 876-6115

After

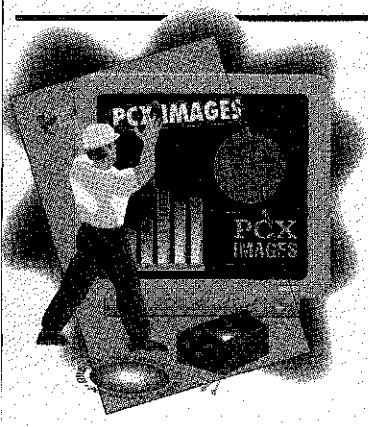
Call about Custom Device Driver Development Services

Circle 82 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADD THE POWER OF GRAPHICS TO YOUR PROGRAMS WITH THE GENUS GX DEVELOPMENT SERIES

PCX PROGRAMMER'S TOOLKIT 4.0



Finally, a programmer's toolkit that helps you incorporate graphics into your programs quickly and easily. The PCX Programmer's Toolkit includes over 75 routines to display, save, scale, and print PCX bitmapped graphics from almost any program. Includes nine utility programs for capturing screens, displaying images, managing image libraries, and printing image files. Use graphics from PC Paintbrush, Scanners, Fax Machines, Clip Art, or captured screens.

ONLY $249

PCX TEXT

Now you can display blazing bitmapped text in any graphics mode as simply as text mode. With PCX Text, you can display your textual information right along with your graphical data. Routines are included for displaying any font in the GEM font file format, with dynamic attributes like bolding, underlining, transparency, and more. A convenient dual font editor allows for creating your own text styles, or use any of the 70 or more fonts provided.

ONLY $149

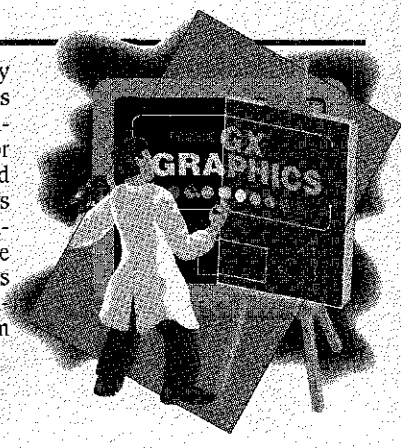
PCX EFFECTS*

Add spectacular special effects (F/X) to your programs, without being trapped in someone else's editor or slideshow program. Wipe, split, crush, slide, sand, drip, diagonal, spiral, random, or explode your graphics. The PCX Effects package is the first to offer professional graphic effects to developers writing their own applications. Jazz up your programs for attention getting results. Includes a complete music definition language.

ONLY $99

* Requires PCX Toolkit

GX GRAPHICS



A complete graphics library supporting all graphics primitives. The GX Graphics toolkit features full support for logical operations, clipping, and drawing to off-screen virtual buffers in conventional memory or expanded (LIM 4.0). The cornerstone of the GX Series, use this graphics library instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers – while supporting more video modes.

ONLY $199

SPECIAL!

SAVE 15% ON ALL FOUR PRODUCTS BY ORDERING THE GENUS GX SERIES DEVELOPER'S PAK FOR ONLY

$549

THE GENUS GX DEVELOPMENT SERIES toolkits are hand coded in Assembly Language and are designed to work together or independently, with no code overhead. Collectively they save you time and money when adding graphics to your programs. GENUS GX SERIES toolkits come with complete instructional manuals and a technical support team to answer your questions. The GENUS GX SERIES supports 12 compilers for C, Pascal, Basic, Fortran, Assembly, and Clipper; over 30 resolutions up to Super VGA, for the fastest, tightest code. Assembly Language source code is available optionally. And each toolkit in the GENUS GX SERIES comes to you ROYALTY-FREE, with a COMPLETE 30 DAY MONEY BACK GUARANTEE.

TO ORDER ANY OR ALL OF THE GENUS GX DEVELOPMENT SERIES, CALL TOLL-FREE:

1-800-227-0918

VISA COD CHK PO

G·E·N·U·S MICROPROGRAMMING

11315 MEADOW LAKE • HOUSTON, TX 77077 • (713) 870-0737

FAX (713) 870-0288 • BBS (713) 266-9362

VA MEMBER Better Business Bureau

MEMBER Software Publishers Association

Circle 261 on Reader Service Card

© 1990 GENUS MICROPROGRAMMING, INC. MANUAL# 4170

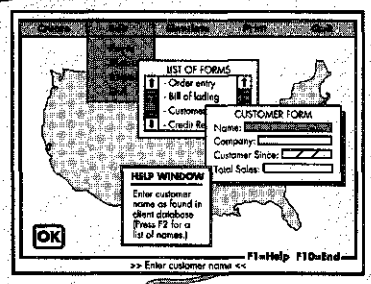
PRODUCED 2006 BY SIMZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

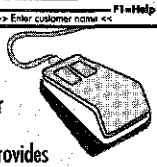
HI-SCREEN XL™

THE PROVEN USER-INTERFACE SOLUTION FOR YOUR PROGRAMS

Since 1983, Softway has specialized in user-interface development tools aimed at demanding professional programmers.

WINDOWS • MENUS DATA ENTRY • HELP





HI-SCREEN XL offers a high-level modular approach to screen management that provides for superior programming **control, flexibility** and **speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules, CGA, EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.

WITH MOUSE AND GRAPHICS SUPPORT

- Supports all versions of **C, Basic, Cobol, Fortran, Pascal, dBase, Assembly,** etc.
- Use linkable (with C and MS Basic) or removable TSR screen driver (30-40K).

HI-SCREEN XL **$149**
HI-SCREEN XL *Professional Series*
(with mouse & graphics support): **$325**

30 day money-back guarantee • No royalties

Call today: **1-800-338-2852**
Tel.: 415-896-0708 • Fax: 415-896-0709

SOFTWAY; Inc.
185 Berry Street, Suite 5411, San Francisco, CA 94107

Circle 110 on Reader Service Card

cycled cleanly, and the other 224 colors would stay the same or would change only occasionally.

That suggests a second possibility: if you have several different color sets to be cycled, you can interleave the loading so that only one color set is cycled per frame. Suppose you are animating a night scene, with stars twinkling in the background, meteors streaking across the sky, and a spaceship moving across the screen with its jets flaring. One way to produce most of the effects with little effort would be to draw the stars in several attributes and then cycle the colors for those attributes, draw the meteor paths in successive attributes, one for each pixel, and then cycle the colors for *those* attributes, and do much the same for the jets. The only remaining task would be to animate the spaceship across the screen, which is not particularly difficult. I'd love to put that demo program in this column, but it would take up an awful lot of space and I'm running out of room here with things left to cover.

ASSIGNING TASKS

The key to getting all the color cycling to work in the example above, however, would be to assign each color cycling task a different part of the DAC, with each part cycled independently as needed. If the total number of locations cycled proved to be too great to manage in one frame, you could simply cycle the colors of the stars after one frame, the colors of the meteors after the next, and the colors of the jets after yet another frame, then go back around to cycling the colors of the stars. By splitting up the DAC this way and interleaving the cycling tasks, you can perform a great deal of seemingly complex color animation without loading very much of the DAC during any one frame.

FLIPPING PAGES

Yet another and somewhat odder workaround is that of using only 128 DAC locations and page flipping. (Page flipping in 256-color modes involves using the VGA's undocumented 256-color modes; see this column in *PJ* 7.1 or my book *Power Graphics Programming* for details.) In this mode of operation, you'd first display page 0, which is drawn entirely with colors 0-127. Then you'd draw page 1 to look just like page 0, except that colors 128-255 are used instead. You'd load DAC locations 128-255 with the next cycle settings for the 128 colors you're using, then switch to display the second page with the new colors. You could then modify page 0 as needed, drawing in colors 0-127, load locations 0-127 with the next color cycle settings, and flip back to page 0.

The idea is to modify only those DAC locations that are not used to display any pixels on the current screen. The advantage is not, as you might think, that you don't generate garbage on the screen when modifying undisplayed DAC locations; in fact, you do. A spot of interference will show up if you set a DAC location, displayed or not, during display time. You still have to wait for vertical sync and load only during non-display time before loading the DAC when page flipping with 128 colors.

The advantage is that since none of the DAC locations you're modifying is currently displayed, you can spread the loading out over two or more vertical non-display periods—however long it takes. If you did this without the 128-color page flipping, you might get weird on-screen effects as some of the colors changed after one frame, some after the next, and so on, or you might not; changing the entire DAC in chunks over several frames is another possibility worth considering.

Yet another approach to color cycling is to load a bit of the DAC during each horizontal non-display period. Combine that with counting scan lines, and you could vastly expand the number of simultaneous on-screen colors by cycling colors as a frame is displayed, so that the color set changes from scan line to scan line down the screen.

The possibilities are endless. However, were I writing 256-color software that used color cycling, I'd find out how many colors could be cycled after the start of vertical sync on a 4.77 MHz PC (or the slowest computer I expected the software to run on), I'd lop off at least 10% for a safety margin, and I'd structure my program so that no color cycling set exceeded that size, interleaving several color cycling sets if necessary.

That's what *I'd* do. But don't be held back by my limited imagination! Color cycling may be the most complicated of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        stosw                           ; them to the PaletteTemp
        mov     es:[di],bl              ; location for attribute 255
if WAIT_VSYNC
;Wait for the leading edge of the vertical sync pulse; this ensures
; that we reload the DAC starting during the vertical non-display
; period.
        mov     dx,INPUT_STATUS_1
WaitNotVSync2:                  ;wait to be out of vertical sync
        in      al,dx
        and     al,08h
        jnz     WaitNotVSync2
WaitVSync2:                     ;wait until vertical sync begins
        in      al,dx
        and     al,08h
        jz      WaitVSync2
endif   ;WAIT_VSYNC
if USE_BIOS
;Set the new, rotated palette.
        mov     ax,1012h        ;AH = 10h: set DAC function,
                                ; AL = 12h: set DAC block subfunction
        sub     bx,bx           ;start with DAC location 0
        mov     cx,CYCLE_SIZE   ;# of DAC locations to set
        mov     dx,seg PaletteTemp
        mov     es,dx
        mov     dx,offset PaletteTemp ;point ES:DX to array from which
                                ; to load the DAC
        int     10h             ;load the DAC
else    ;!USE_BIOS
 if GUARD_AGAINST_INTS
        mov     cx,CYCLE_SIZE   ;# of DAC locations to load
        mov     si,offset PaletteTemp ;load the DAC from this array
        sub     ah,ah           ;start with DAC location 0
DACLoadLoop:
        mov     dx,DAC_WRITE_INDEX
        mov     al,ah
        cli
        out     dx,al           ;set the DAC location #
        mov     dx,DAC_DATA
        lodsb
        out     dx,al           ;set the red component
        lodsb
        out     dx,al           ;set the green component
        lodsb
        out     dx,al           ;set the blue component
        sti
        inc     ah
        loop    DACLoadLoop
 else   ;!GUARD_AGAINST_INTS
        mov     dx,DAC_WRITE_INDEX
        sub     al,al
        out     dx,al           ;set the initial DAC location to 0
        mov     si,offset PaletteTemp ;load the DAC from this array
        mov     dx,DAC_DATA
  if NOT_8088
        mov     cx,CYCLE_SIZE*3
        rep     outsb           ;load CYCLE_SIZE DAC locations at once
  else  ;!NOT_8088
        mov     cx,CYCLE_SIZE   ;# of DAC locations to load
DACLoadLoop:
        lodsb
        out     dx,al           ;set the red component
        lodsb
        out     dx,al           ;set the green component
        lodsb
        out     dx,al           ;set the blue component
        loop    DACLoadLoop
  endif ;NOT_8088
 endif  ;GUARD_AGAINST_INTS
endif   ;USE_BIOS
;See if a key has been pressed.
        mov     ah,0bh          ;DOS check standard input status fn
        int     21h
        and     al,al           ;is a key pending?
        jz      CycleLoop       ;no, cycle some more
;Clear the keypress.
        mov     ah,1            ;DOS keyboard input fn
        int     21h
;Restore text mode and done.
        mov     ax,0003h        ;AH = 0: set mode function,
        int     10h             ; AL = 03h: mode # to set
        mov     ah,4ch          ;DOS terminate process fn
        int     21h
        end     start
```


INTEL DEVELOPMENT TOOLS

For the ultimate success with your microcomputer development project Intel offers you the total solution. From emulators to compilers to performance analysis and simulation, Intel can supply you with the most up-to-date and powerful development tools available today.

We now offer you an easy method of ordering. Simply call our toll-free number at 1-800-874-6835 to request a complimentary copy of our Intel Development Tools Catalog listing all our offerings. Or drop us a line at: Intel Corporation, Catalog Sales, 5200 NE Elam Young Pkwy, JF1-15, Hillsboro, OR 97124. After you have had a chance to look over our products, give us a call. We can answer any questions you may have about Intel Development Tools and we'll be happy to take your order over the phone. No waiting. It's as simple as that. So call us today. We'll be waiting to hear from you.

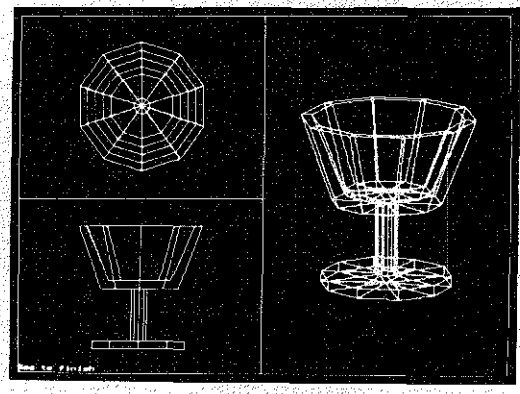
CALL US TODAY 1-800-874-6835

Circle 50 on Reader Service Card

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Circle 229 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SLATE Universal Printer Driver

NEW! Text & Graphics!

By including SLATE, you can print both **Text and Graphics** on **over 400 printers. Immediately! Painlessly!**

You can use SLATE in your product with **no royalties.**

Make your product more functional and competitive by taking advantage of SLATE's **advanced text features:**

- Support multiple printers on the same system.
- Output to parallel printers, serial printers, DOS files, console, DOS print spooler, and Novell network printers.
- Support proportional fonts laser printer soft fonts.
- Set exact print positions.

You can add the **NEW** Graphics Subsystem for **advanced graphics features:**

- Print images from the screen, graphic files, or custom image systems.
- Scale and Rotate the printed image.
- Convert colors to patterns.
- Print grey scale (shaded) and color images.
- Intermix text and graphics.

What is SLATE?

SLATE is a set of C libraries with over 150 text printing functions, a Database of over 400 printers, and tools for end user configuration and testing.

SLATE with Graphics adds over 60 graphic printing functions.

Call now for more information. Order **SLATE for $299 or SLATE with Graphics for $448** with our risk free, 30 day return policy.

We accept Visa, Mastercard, COD's or PO's from qualified companies. Source code, maintenance, and site licenses are available. Also ask for information about our S_PRINT Text Formatting System for Software Developers.

800-316-3988

Inc. PO Box 20162 Columbus, OH 43220 614-451-1865 Fax 614-459-1925

Circle 81 on Reader Service Card

all the color control techniques, but it's also the most powerful.

ODDS AND ENDS

I'm way over my word budget now, so I'm going to go fast. Watch closely or you might miss something!

In my experience, when relying on the autoincrementing feature while loading the DAC, the Write Index register wraps back from 255 to 0, and likewise when you load a block of registers through the BIOS. So far as I know, this is a characteristic of the hardware, and should be consistent. Also, Richard Wilton documents this behavior for the BIOS in the VGA bible, *Programmer's Guide to PC and PS/2 Video Systems* (Microsoft Press), so you should be able to count on it. Not that I see that DAC index wrapping is especially useful, but it never hurts to understand exactly how your resources behave, and I never know when one of you might come up with a serviceable application for any particular quirk.

THE DAC MASK

There's one register in the DAC that I haven't mentioned yet, the DAC Mask register, at 03C6h. The operation of this register is simple but powerful: it can mask off any or all of the 8 bits of pixel information coming into the DAC from the VGA. Whenever a bit of the DAC Mask register is 1, the corresponding bit of pixel information is passed along to the DAC to be used in looking up the RGB triplet to be sent to the screen. Whenever a bit of the DAC Mask register is 0, the corresponding pixel bit is ignored, and a 0 is used for that bit position in all look-ups of RGB triplets. At the extreme, a DAC Mask setting of 0 causes all 8 bits of pixel information to be ignored, so DAC location 0 is looked up for every pixel, and the entire screen displays the color stored in DAC location 0. Incidentally, setting the DAC Mask register to 0 is a quick and easy way to blank the screen.

READING THE DAC

I've been promising to tell you how to read the DAC for nearly half a year now, so I suppose I'd better squeeze it in. The DAC can be read directly via the DAC Read Index register at 3C7h and the DAC Data register at 3C9h. Reading the DAC directly works in much the same way as writing it directly by way of the DAC Write Index register, complete with autoincrementing the DAC Read Index register after every three reads. Everything I've said about writing the DAC applies to reading the DAC. In fact, reading the DAC can even cause snow, just as loading the DAC does, so it should ideally be performed during vertical non-display time.

The DAC can also be read via the BIOS in either of two ways. INT 10h, function 10h (AH=10h), subfunction 15h (AL=15h) reads out a single DAC location, specified by BX. This function returns the RGB triplet stored in the specified location, with the red component in the lower 6 bits of DH, the green component in the lower 6 bits of CH, and the blue component in the lower 6 bits of CL.

INT 10h, function 10h (AH=10h), subfunction 17h (AL=17h) reads out a block of DAC locations of length CX, starting with the location specified by BX. ES:DX must point to the buffer in which the RGB values from the specified block of DAC locations are to be stored. The form of this buffer (RGB, RGB, RGB...) is exactly the same as the form of the buffer used when calling the BIOS to load a block of registers.

Listing 1 illustrates reading the DAC via the BIOS block-read function and reading it directly, with the direct-read code capable of conditionally assembling to either guard against interrupts or not and to either use **rep insb** or not. As you can see, the process of reading the DAC settings is symmetric to that of setting it.

THE FINAL REWARDS

And so, at long last, we come to the end of our discussion of color control on the VGA. If it has been more complex than anyone might have imagined, it has also been more rewarding; there's as much obscure but very real potential to color control as there is anywhere on the VGA, which is to say that there's a very great deal of potential indeed. Put color cycling or color paging together with the page flipping and image drawing techniques we've explored over the last four years, and you'll leave the audience gasping and wondering, "How the heck did they do that?" PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Develop Powerful Image-based Applications Quickly and Easily

with

Victor Image Processing Library

Powerful Image Processing

VICTOR simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you powerful image processing (brighten, alter contrast, sharpen, outline, linearize, matrix convolution, overlay, resize, flip, mirror, and much more), screen display, file handling, and the ability to hold multiple images in expanded memory.

Image-based applications can be developed in QuickC and Turbo C environments.

Gray Scale and Color Images

Victor allows you to handle 8-bit color and gray scale images from any source, as TIFF/PCX/GIF files, and display single or multiple images on EGA/VGA up to 1024x768x256. Your software can have features such as: zoom, resize, display multiple images, pixel editing, image processing, and laser printing. And, to get you up and running, we've included Zip Image Processing software for rapid testing and prototyping of image processing and display functions.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available.

Your application will be able to print halftones at any size.

Victor Library $195

Call (314) 962-7833 to order

VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 962-7833

Circle **321** on Reader Service Card

MORE 34010 Video Co-Processors from PC Tech

- Mono 736 x 1024 - 2048 x 1536
- Color 1024 x 768 (1280 x 1024 soon!)
- 512K - 4M DRAM
- Optional CCITT & DGIS support
- **AGE XoftWare X Window Server certified board**
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- **TIGA** included with all boards
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PTech

907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, XoftWare, and DGIS are trademarks or registered trademarks of their respective companies.

Circle **208** on Reader Service Card

SPOTLIGHT ON SUCCESS:

Our goal was to create a comprehensive mine planning system for our underground mining operations. Using SilverScreen's C compiler we have developed a mine planning system that uses SilverScreen as its CAD platform. Our system is two years ahead of others like it.

Peter S. Campbell
Mines Computer Systems
Senior Project Engineer
Falconbridge, Limited

CAD ENGINE

SilverScreen Licensing Program

SilverScreen is a 3D CAD and Solid Modeling software system that contains a resident C Language compiler. The C compiler was designed by Schroff Development for use by end users and software developers to facilitate the development of custom applications requiring 3D and solid modeling.

SilverScreen is the most powerful and versatile 3D CAD/Solid Modeling system available containing such features as: 2D and 3D Boolean operations; rendering; shading; mass properties; text editor; associative dimensioning (ANSI Standard); object-oriented database; schema-based attributes; and camera walk. SilverScreen runs on DOS and UNIX (Silicon Graphics) and supports a wide range of mice, video boards, printers, plotters and digitizers. File transfer is handled through DXF, IGES, HPGL, PostScript and others. Silverscreen features direct export to SmartCAM and Stereolithography (3D Systems).

SilverScreen's resident C compiler implements a library of more than 300 functions plus a large number of functions dedicated to accessing the SilverScreen database (from which parts lists and bill of materials can be generated) and to controlling the SilverScreen environment. Custom menus and icons also may be created. SilverScreen also adds the powerful ability to embed the SilverScreen script commands in a program for direct execution by SilverScreen.

For SilverScreen information contact:

Schroff Development Corporation
P.O. Box 1334
Mission, KS 66222
(913) 262-2664

Dealer Inquiries invited

Evaluation packages available

SilverScreen®

3D CAD/Solid Modeling Software

Circle **65** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Digital typography

Font & type generation for graphics screens and laser printers

By Ron Fosner

Digital typography is rapidly becoming a hot issue. Inexpensive laser printers have made the quality of a program's output a matter of increasing importance. When the first HP Laserjet appeared, users were ecstatic over the quality of type and the sharpness of both text and graphics. Most people were not bothered by the fact that there was little control over the textual output. This euphoria lasted until Apple brought out the LaserWriter in 1985. Users then discovered PostScript, along with a previously unsuspected realm of assorted typefaces in multiple sizes.

Now, the era of desktop publishing has truly arrived. Programs are no longer judged solely on how fast they perform various operations and how good their documentation is; they must also be able to produce output of a quality that would have been unthinkable just five years ago.

As soon as you allow the user to set printer-specific codes to generate bold, italic, or proportional text, supporting any printer becomes much more complex. Software engineers must now cope with the fact that it's no longer acceptable to simply send a string of characters to LPT1. This dilemma has three parts: allowing the user to see on the screen any font their printer can produce (or a reasonable facsimile); finding out which fonts their printer can support; and generating fonts that are not supported by the printer.

Ron Fosner is a senior engineer at Lotus Development who started out as a chemical engineer. He likes to solve engineering problems using software, but will take on all interesting problems. Contact him at Lotus Development Corp.-3 North, 1 Rogers Street, Cambridge, MA 02141.

FONT INDEPENDENCE

Since each printer is different, each needs a different printer driver. These drivers need to know not only how to display text and graphics, but also what fonts are available in the printer. This knowledge does not address the problem of letting the user select typefaces or font sizes other than the supplied ones. However, it seems foolish to run a WYSIWYG (What You See Is What You Get) program on a SuperVGA screen and have the screen font quality be limited to a simulation of the output of the draft-mode, 9-pin dot matrix printer that happens to be attached to the computer. The solution lies in finding a font generation method which is independent of any printer limitations.

While we're generating fonts for the screen, we can also generate them for the printer. We can display any font we wish if we generate the appropriate bitmap by treating the printer as a raster device. This article illustrates how to take a font file and hook up the font vendor's character-generating routines to let your program make almost any size font for a variety of display devices. You can provide true WYSIWYG capabilities for your program this way, while freeing yourself of the constraint of displaying only those fonts available from the printer or video board.

Font independence gives you the ability to update or add new fonts. By using a "standard" font interface (although there is more than one "standard" at the moment), your programs can see what fonts are available and allow users to access them in their applications. Adding a new font should require nothing more than providing some way of allowing the user to specify a particular font file. For the user, this means simply copying the font from the floppy disk sent by the manufacturer into a font directory, firing up any application that uses the same font format, and then selecting the newly loaded font.

FONTS ON THE FLY

Most WYSIWYG programs present fonts via video bitmaps that correspond to the fonts available from the attached printer. This limits the user to the printer's native fonts. Printing the same layout on another printer means the program has to do some guessing so that what you see on the screen is as close as possible to what you get as printed output. A method of generating the desired font

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

on the fly would free us from having to limit the output to the printer's resident fonts or to ones that are similar.

In addition to character bitmaps, most font software packages allow the generation of the actual outlines of characters in terms of line segments, arcs, and/or Beziér curves. Since we are generally writing to pixel devices, I'll discuss font generation only in terms of character bitmaps. Keep in mind that everything that is said about bitmaps will usually hold true for character outlines as well.

As of this writing, only two font vendors will supply source code—Bitstream and AGFA Compugraphic (see Font Vendors box for addresses). Bitstream fonts are bundled with a number of DOS word processors, including Word Perfect, WordStar, and Microsoft Word. AGFA's scalable typefont technology is incorporated in the HP Laserjet III printers and PCL-5.

The first end-user font libraries were Bitstream's Fontware and Compugraphic's Intellifont, both available in 1986. In 1989, Bitstream introduced Speedo and Compugraphic came out with Bullet. These updated versions have faster character generation and reduced code size. Best of all, both are inexpensive and easy to incorporate into an application. Font vendors realize that the easier it is for applications to use their fonts, the more their fonts will be used.

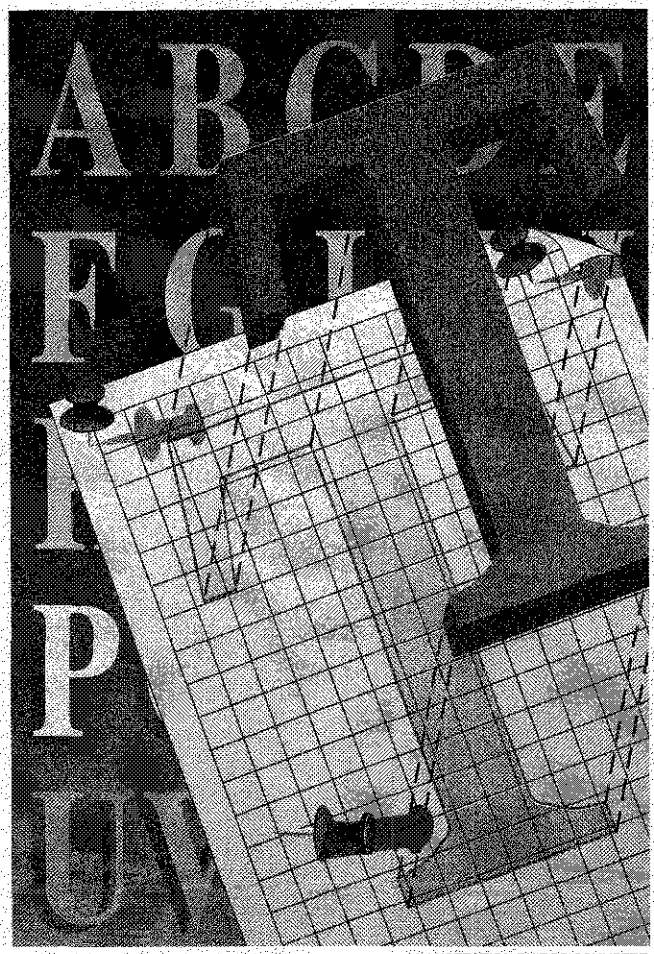


ILLUSTRATION BY ROB SCHUSTER

AN OVERVIEW OF SPEEDO

The technology used in this article is Bitstream's Speedo font generation package. This is due to my familiarity with Speedo and its simple interface, not an endorsement of the product specifically. Because Speedo's layout is similar to that of other font generation packages, what is described for Speedo can be generalized to other packages.

The Speedo subroutine library comes in a three-ring binder with fairly complete documentation, program sources, sample font files, and some demo and utility programs. To use the Speedo library you simply follow these six steps:

1. Initialize it with a function call.
2. Open the font file for the typeface.
3. Set the specifications for the font.
4. Call a library routine to generate the scaled outlines.
5. Call a library routine to generate the desired character bitmap or outline.
6. Close the file.

If you want different sized or transformed characters in the same typeface, repeat steps 3-5. If you want an entirely new typeface, repeat steps 2-5.

Sounds simple, doesn't it? You'll discover that there's a bit more bookkeeping involved, since you'll have to keep track of where the text gets displayed and where the current position is in order to start drawing the next character. The characters you're now displaying are actually graphics objects that can be almost any size. Each character comes with its own information about how tall and wide it is, where the character following it should start, and the dimensions of its bitmap matrix, along with the actual bitmap. The display of lines of text is a bit more complicated, but the stunning results make it well worth the effort.

HOOKING UP SPEEDO

A complete description of Speedo would be too large to include in this article. I will, however, cover its basic library functions and what they do. Before using the library, you need to create a few specific subroutines. Once these have been provided, using the library is quite simple.

The function **sp_reset(void)** must be called once to initialize the Speedo subsystem. This prepares the library routines for the actual font generation and is called before any other routine in the library.

After the Speedo font file has been opened and the font information read in, the **sp_set_specs (spects_t *pspecs)** function sets or modifies the current font characteristics. The pointer that is passed contains various parameters describing the font to be generated as well as pointers to the information read in from the file. It is possible to read in a minimal amount of the file and let internal routines read in more as needed. The sample program in Listing 2 shows how this is done.

GENERATE AND DISPLAY

The function **sp_make_char (ufix16 index)** tells the library to generate the character bitmap. The "ufix16" tag is an unsigned short typedef, and "index" is the number of the character to be generated. The lower 126 values correspond to their ASCII equivalents. At that point, **sp_make_char** calls up three functions which you will need to supply in your program: **sp_open_bitmap**, **sp_set_bitmap_bits**, and **sp_close_bitmap**.

The first and last of these functions signal the start and completion of the bitmap, while the middle is called for each line of pixels to be set in the bitmap. It is the calling program's responsibility to actually do something with the information provided.

In the sample program, a call is made to **sp_make_char** each time a character

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FONT VENDORS

AGFA Compugraphic Division
Agfa Corporation
90 Industrial Way
Wilmington, MA 01887
(508) 658-5600
Contact: OEM Technical Support Manager

Bitstream Inc.
Athenaeum House
215 First St.
Cambridge, MA 02142
(617) 497-6222
Contact: Strategic Alliance Group

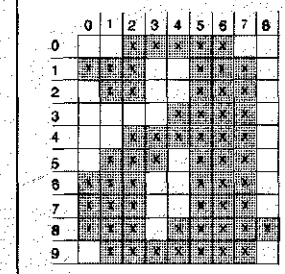
is required. Calls to these three functions are then used to write the character to the screen. When **sp_set_bitmap_bits** is called by the library, it receives raster line information about the character bitmap. Each of these calls contains the horizontal row number and the starting and ending column number.

Until all of the raster lines have been sent, **sp_set_bitmap_bits** will be called repeatedly. Figure 1 is an example of a character grid with the corresponding parameters from a series of calls to **sp_set_bitmap_bits** for generating a lowercase "a."

Normally, smaller characters would be generated once and their bitmaps would then be placed in a character cache so they would be immediately available. But in certain cases, such as low memory or large character generation, the character bitmaps would only be generated as needed. In order to keep Listing 2 fairly simple, the character display routines draw the raster lines as they come in to **sp_set_bitmap_bits**.

When the character is finished, there will be a call to **sp_close_bitmap** which is an indication to update the current character position in order to be ready for the next character. This basically means incrementing the x and y positions from the values initially passed by **sp_open_bitmap**. Special characters must be handled separately. For newlines, for example, move the x position to the original location and the y position down by the point size.

FIGURE 1



CUSTOMIZING THE TEXT

Horizontal and vertical size (controlled by *xxmult* and *yymult*) are just two of the six values that specify how the generated characters are to look. Two other parameters, *xoffset* and *yoffset*, specify the horizontal and vertical offset of the

Maximum Software Security

100% Surface Mount

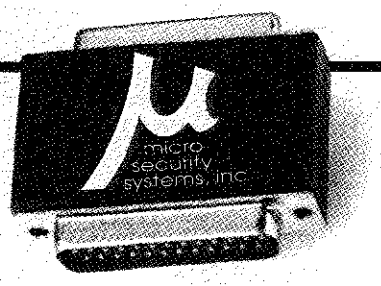
New—Macintosh Version

SecuriKey is a hardware device designed to prevent unauthorized use and duplication of computer software.

- Security leader with 8 different devices
- Works on all IBM/compatibles and Macs
- Transparent to printer
- Allows unlimited back-up copies
- Protects domestic & intern'l sales profits

Call Today! (800) 456-2587

Micro Security Systems Inc. 4750 Wiley Post Way, Suite 180
Salt Lake City, Utah 84116 (801) 575-6600 FAX (801) 575-6621



Circle 220 on Reader Service Card

We Have Ways To Increase Your Productivity

TE Developer's Kit ($139)

Why reinvent the wheel? TE Developer's Kit provides the most cost effective way of incorporating text editing features into your application. The kit includes the complete source code for the editor and the executable file. Supports Microsoft and Borland 'C' compilers.

Features: multiple files/windows, word-wrap and printing, edits files larger than available memory, reconfigurable keyboard commands and screen colors, keystroke macros, undo, column block commands, search/replace, and autosave. The kit also includes TER small editor routine. TER is a subset of TE. Your application calls this routine by specifying the window location and size, maximum file size, and an input buffer or file.

WinMem ($95): Windows 3 Memory Manager

WinMem provides an unlimited amount of virtual memory even in Windows' Real and Standard modes. The device independence is not enough. WinMem offers you CPU and memory resource independence. It provides an unlimited number of global memory handles, and reduces the overhead per block to 4 bytes (from 20 bytes of Windows). WinMem will trap any attempt to free an illegal memory block. This feature alone can save you a countless number of hours of debugging memory problems. Includes the complete source code.

Team ($119): The Programmer's Text Editor

An advanced text editor for a not so advanced price. Features: edits unlimited size files, unlimited levels of undo/redo, multiple zoomable windows, regular expression search/replace, editable keystroke macros, 'C' like macro language, task switching, customizable context sensitive help, customizable keyboard commands, menu with accelerator keys, frees entire memory for DOS commands, 132 column screen mode, and much more.

Sub Systems, Inc. 1-800-447-6819
Fax: 1-617-438-8901
159 Main Street, #8C, Stoneham, MA 02180, 1-(617)-438-8901

Circle 68 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FONT TERMINOLOGY

Typography has its own unique terminology. I have provided definitions of a few of the more common phrases that you might run across:

- **Point size**: This commonly-used term refers to a system of measurement that dates back to the days of actual metal type. The term has held over into the age of digital typography. A point is about 1/72nd of an inch, meaning that a 12-point font would have a distance between baselines of about 1/6th of an inch. Note that the actual character height would be smaller than 12 points. When comparing font sizes, it is usually the x-heights (defined below and illustrated in Figure 2) that are examined, rather than the point sizes.
- **Typeface**: This is the name given to a character set that has specific shapes created by the typographer. Century, Times Roman, Gothic, Courier, and Helvetica are examples of typefaces. Century Bold and Century Italic are different typefaces than Century, because they are typefaces which were created by a typographer. Typically, typefaces come in families of four: normal, bold, italic, and bold-italic. Some font-generation programs can create pseudo bold and italic effects, which tends to confuse things.
- **Font**: A font is a typeface of a particular size. Times Roman Italic is a typeface, 12-point Times Roman Italic is a font. In common usage, you will often see the word "font" used when "typeface" is meant.
- **Serif and Sanserif**: A serif is a small finishing stroke. Serifs add to the readability of a font, especially in the smaller font sizes. Times Roman is a serifed font, and Helvetica is a sanserif (without serifs) font.
- **x-height**: The height of the lowercase letter "x" in a font, the x-height is a useful measure of how the font looks. Note that fonts of similar x-height need not have the same point size. In most cases, the larger the relative x-height, the easier the typeface is to read.
- **Em square**: This is another term from the days of metal type. In 12-point type, an em square would be 12 points wide and 12 points high.
- **Kerning**: This refers to the shifting of intra-letter spacing so that there is no excess space in a word. For example, the lowercase "o" in the kerned word "Toy" is shifted under the "T," while the tail of the "y" is shifted under the "o."
- **Hints**: This is the most important part of font technology. Each vendor has what is usually a proprietary set of algorithms for making bitmapped characters look good. These sets are called the "hints." You'll find that hints become very important when you're generating bitmaps for small characters on medium- to low-resolution devices. Hinting makes sure that things like stem widths are uniform across each of the characters in the font, that all serifs are displayed, and that other details which greatly contribute to the readability of the font are correct and consistent.

LISTING 1

```
/*      LOCAL.H       */
#include <conio.h>                  /* for kbhit prototype */
#define WAIT_FOR_KEY        { while( !kbhit ); getch(); }
/* some of graphics library specific code is separated here */
#ifdef MSDOS                        /* Microsoft C Compiler */
#define ERASE_SCREEN        _clearscreen( _GCLEARSCREEN )
#define RESET_VIDEO_MODE    _setvideomode( _DEFAULTMODE )
   /* for VGA screen */
#define COLORS   16
#define NUM_ROWS 640
#define NUM_COLS 480
#define VIDEO_MODE  _VRES16COLOR
#else       /* define these for your own graphics library */
#define ERASE_SCREEN
#define RESET_VIDEO_MODE
#define WAIT_FOR_KEY
#endif

typedef unsigned int UINT;
typedef unsigned long ULONG;

struct VIDEO {
    UINT x_pixels, y_pixels, colors, mode;
    } video;
typedef struct {
    UINT x, y;
    } POINT;

void origin( void );
void newline( void );
char * start_speedo( char * );
void place_text( char * );
```

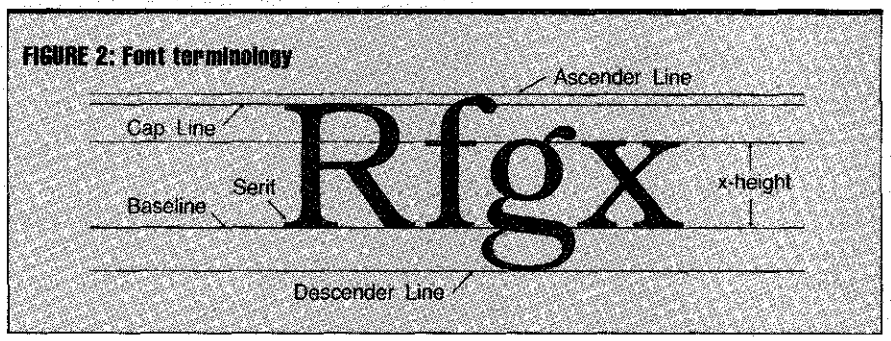
LISTING 2

```
/* FONTDEMO.C - A program to demonstrate dynamic font generation. */
#include   <stdio.h>
#include   <stdlib.h>
#include   <graph.h>
#include "local.h"
/* dimensions of typical screen - video origin is upper left */
#define X_RESOLUTION 10             /* 10.0 inches  */
#define Y_RESOLUTION 7.5            /*  7.5 inches  */
    /* function prototypes */
void main( int argc, char **argv );
void set_video( void );
    /* global data */
int PixelsperInch_x, PixelsperInch_y, Point_x, Point_y,
    Value_xy,Value_yx;
short color0 = 1, color1 = 1, color2 = 1;
POINT points_per_pixel;

/* main: called with one argument that is the location of the
 * Bitstream Speedo font file name. It then loops though a collection
 * of settings for the font, displaying a few lines of text each time,
 * while waiting for a key press before proceeding. */
void main( int argc, char **argv )
{
    char * mssg;
    int i;
    /* the following are the parameters for the font calls */
    static int size = 6;
    static int params[ ][6] = {
        { 24, 24,  0,  0},
        { 72, 48,  0,  0},
        { 72, 72,  0,  0},
        { 72, 72, 12,  0},
        { 72, 72,  0, 12},
        { 72, 72, 12, 12},
    };
    if ( argc != 2 ) {
        printf("Need to specify a font file\n");
        exit( 1 );
    }
    /* reset text drawing information, get video information */
    set_video();
    for ( i = 0 ; i < size ; i++ ) {
        ERASE_SCREEN;  /* erase entire screen */
        /* set the Speedo parameters to new values */
        Point_x  = params[i][0];
        Point_y  = params[i][1];
        Value_xy = params[i][2];
        Value_yx = params[i][3];
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2: Font terminology



character. In this program we do the bookkeeping ourselves, since it folds in nicely with the information about where the text originally starts and how much to move the origin for a new line of text.

Figures 3 and 4 are examples of the effects of the values *xymult* and *yxmult*. The *xymult* parameter allows you to specify the amount of obliqueness, so the angle of the character's vertical baseline shifts according to the degree of obliqueness set by *xymult*, providing a pseudo-italic effect. It can also be used for special effects such as extreme or reverse slants. The last parameter, *yxmult*, lets you modify the horizontal baseline as a function of the vertical position. Using *yxmult* alone pulls the characters vertically; used with *xymult*, it makes rotated characters.

These six values generate characters by using actual hinted bitmaps. These bitmaps have been specifically generated to be displayed at the specified size, rotation, and angle—they are not bitmaps which have simply been rotated. The extra work is amply rewarded by the result—high-quality characters that look as good as any that can be generated.

READY FOR THE FONTS

Listing 3 contains the non-library-specific routines. Before you call into the library, you need to specify the parameters of the fonts to be generated. The graphics resolution of the display is determined by the values of the global variables *PixelsperInch_x* and *PixelsperInch_y*. Since most fonts are specified in point values, you need to know how pixels relate to points.

There is no way to ask the hardware for the vertical and horizontal dimensions of the physical screen, so you'll find the constants *X_RESOLUTION* and *Y_RESOLUTION* set to 10 and 7.5 inches, respectively, for the number of inches I measured on my particular video monitor. An alternative is to use the resolution of the device (for instance, a 300 dpi laser printer), but you'll need either a table of devices or some way of setting

"If I could put together a personal suite of programmer's utilities, I would include the C Programmer's Toolbox for a nifty Swiss-Army-Knife collection of utilities."
Warren Keuffel, *Computer Language*, June 1990

The C Programmer's Toolbox™ contains more than 20 tools for the serious C programmer. The tools are available in three different development environments: PC/MS DOS; Macintosh's MPW; and Sun's UNIX. Some of the major tools include:

CDecl	Translates C declaration statements to/from English *(helps you decipher those complicated C variable declarations and construct new ones). {MPW & Sun only}*
CFlow™	Determines how a program is organized and what is in the runtime library *(quickly shows you function hierarchy and interdependancies)*
CHilite™	Highlights and prints your C source code *(great for creating your final documentation). {MPW only}*
CLint™	Syntactically checks one or more of your C source code files & **generates ANSI function prototypes** *(use this to find those latent programming bugs)*
Cpp	Preprocesses C source code files that have ANSI macro definitions *(helps you port ANSI C code to/from your target environment). {MPW & Sun only}*
CPrint™	Reformats/beautifies your C source code *(makes code easy to read and ends disputes on source code formats).*
CritPath™	Determine a program's critical path. *(helps you improve your program's performance) {PC DOS only}*
CXref™	Cross references and checks usage of up to 17 different classes of symbols in your C source code *(cross references any size program and number of listings)*
PMon™	Monitors a program's execution *(non-invasively monitors function and system service execution. {PC DOS only}*

"I recommend getting both (Volumes)... compared to UNIX tools, the Toolbox tools have more options and provide more detailed information..." —*Kenji Hino, The C User's Journal, February 1990*

C™ Programmer's Toolbox
New 2.1 Release
MPW (Macintosh) / PC (PC/MS DOS) / Sun (UNIX)

The tools include support for ANSI C, Microsoft C *(new support for 6.0)*, Turbo C, Sun UNIX C, Macintosh's MPW C, THINK C *(new support for 4.0 OOPs)* and more. The Toolbox is *fast*, handles *any size program* and is easy to use. Several tools also include a C preprocessor that accepts K&R, ANSI and other C dialects. Improve your productivity and your program's performance and quality today:

C Programmer's Toolbox™
MPW ($295), PC Volume I or II ($99.95),
PC Volume I & II ($175), Sun ($495)
C Programmer's McTool Series™ (Macintosh - non-MPW)
McCPrint™ ($99.95), McCLint ($149.95)

"The documentation is exemplary... strong internal design..."
—*Will Watts, EXE, April 1990*

Call and Order today...
(CA residents: please add sales tax; Overseas: contact us for details)

MasterCard
VISA®

MMC AD Systems™
Box 360845
Milpitas, CA 95036
Phone: (415) 770-0858
Fax: (415) 770-0116

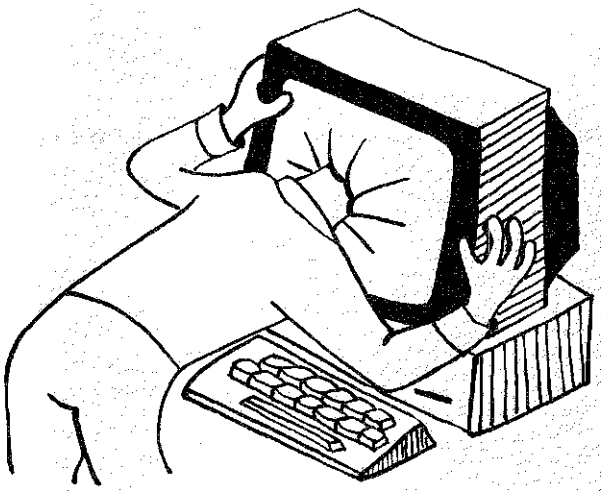
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        /* call Speedo to generate outlines */
        if ( ( mssg = start_speedo( argv[1] ) ) != NULL ) {
            /* reset video mode and print error message */
            RESET_VIDEO_MODE;
            printf("%s\n",mssg);
            exit(1);
        }
        origin();           /* clear screen, reset text origin */
        newline();          /* move down one line from the top */
        place_text("Hello World!");
        newline();
        place_text("Hi Again!\n");  /* note embedded newline */
        place_text("ABCDEabcde12345678901");
        WAIT_FOR_KEY;               /* wait for a keypress */
        }
    RESET_VIDEO_MODE;               /* reset video mode */
}
/* set_video: this routine sets video mode to graphics mode, generates
 * global values for video resolution, sets color values */
void set_video( void )
{   struct videoconfig config;
    if ( _setvideomode( _VIDEO_MODE ) == 0 ) {
        printf("Videomode failed\n");
        exit(1);
    }
    _getvideoconfig( &config );
    /* set the globa data */
    video.x_pixels = config.numxpixels;
    video.y_pixels = config.numypixels;
    video.colors   = config.numcolors;
    if ( video.colors > 2 ) {       /* a color screen */
        color0 = 15;                /* _BRIGHTWHITE; */
        color1 = 1;                 /* _BLUE;        */
        color2 = 1+8;               /* _LIGHTBLUE;   */
    }
    points_per_pixel.x = 72 * X_RESOLUTION / video.x_pixels;
    points_per_pixel.y = 72 * Y_RESOLUTION / video.y_pixels;
    PixelsperInch_x = video.x_pixels / X_RESOLUTION;
    PixelsperInch_y = video.y_pixels / Y_RESOLUTION;
}
```

LISTING 3

```
/* SPEEDO1.C - Demonstrates dynamic font generation.
 *       This file contains the library specific calls & code */
#define PROTOS_AVAIL 1   /* use function prototypes in speedo.h */
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include "speedo.h"     /* include the speedo header */
#include "local.h"      /* local definitions for this program */
        /* static function prototypes */
static ufix8 read_1b(ufix8 *ptr);
static fix31 read_4b(ufix8 *ptr);
static fix15 read_2b(ufix8 *ptr);
static void place_char( char c );
        /* external data used by the output routines */
extern POINT points_per_pixel;
extern int  PixelsperInch_x, PixelsperInch_y,
        Point_x, Point_y, Value_xy, Value_yx;
extern short color0, color1, color2;
        /* static data used by the input routines */
static ufix16  ORUs_per_em;
static ufix8 *font_buffer = NULL;   /* Pointer to allocated font buffer */
static ufix8 *char_buffer = NULL;   /* To allocated Character buffer    */
static buff_t font;                 /* Buffer descriptor for font data  */
static ufix16 minimum_char_buffer_size;   /* Min. character buffer size */
static ufix32 minimum_buffer_size; /* Min. font buffer size to allocate */
static int first_char_index;        /* Index of first character in font  */
static FILE *fp = NULL;             /* the font file pointer */
        /* static data used by the output routines */
/* offsets of where to start the first character */
static POINT char_origin, char_shift, line_origin;

/* origin: Resets the local origin values to their initial values */
void origin()
{   char_origin.x = char_origin.y = 0;
    line_origin.x = Point_x*points_per_pixel.x;
    line_origin.y = Point_y*points_per_pixel.y;
}
/*  newline: Resets the local horizontal origin to the next line
 * (moves it down by the point size ) resets the vertical to its
 * original value, and draws in a baseline */
void newline( void )
{   char_origin.x = char_origin.y = 0;
```

continued



Look inside your software!

Quaid Analyzer shows the inner workings of software, even without the source. Scroll through text or instructions as easily as with an editor.

You can get a report on the screen at every DOS service call, or every interrupt or breakpoint.

Ordinarily **Quaid Analyzer** is memory resident. For demanding tasks, load it before DOS, then watch DOS load and install device drivers. **Quaid Analyzer** includes ways of tracing in spite of countermeasures used to make tracing difficult. We use it to unravel copy protection. **Quaid Analyzer** is pure software, so you don't have to take apart your computer.

To order **Quaid Analyzer**, call us with your credit card, or send us a check for $200 US funds. We ship within a day at our expense.

Quaid Analyzer comes with a manual, and software on a 3 inch and a 5 inch diskette. It is not copy protected. If you are not satisfied, you can return it within 30 days for a refund.

Quaid Software Limited

45 Charles Street East, Third Floor Dept P
Toronto Ontario Canada M4Y 1S2

(416) 961-8243 fax: (416) 961-6448

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    line_origin.x  = Point_x*points_per_pixel.x;
    line_origin.y += Point_y*points_per_pixel.y;
     setcolor( color1 );
     moveto(video.x_pixels-1, line_origin.y);
     lineto(line_origin.x, line_origin.y);
     setcolor( color0 );
}
/* place_char: Takes an ASCII char value, converts it to Bitstream's
 * character index, tells the routine to make the char bitmap (which
 * causes the character to be displayed), then returns. */
static void place_char( char c )
{   ufix16 char_index;    /* Index of character to be generated */
    ufix16 char_id;       /* Character ID */
    if ( c == '\n' ) {
        newline();
        return;
    }
    /* Force Bitstream index to ascii */
    char_index = c + first_char_index - 32;
    char_id = sp_get_char_id((ufix16)char_index);
    /* make the bitmap */
    if ( char_id != 0 )
        sp_make_char(char_index);
}
/*  place_text: Takes a string and calls place_char for each character. */
void place_text( char *string )
{   while ( *string )
        place_char( *string++ );
}
/*  start_speedo:
      Starts the library routines and sets the parameters to
      generate the bitmaps we want. The argument is the location and
      name of the speedo file for the desired typeface. If an
      error occurs, a char pointer is returned pointing to the error
      message, or else NULL is returned. */
char *start_speedo( char *pathname )
{   int bytes_read;
    specs_t specs;   /* Bundle of character generation specs  */
    ufix8 header_buffer[500];    /* Font buffer  */
    char text_string[128];       /* display info */
    fp = fopen (pathname, "rb"); /* Load Speedo outline file */
    if ( fp == NULL )
        return("***** Error: Cannot open the font file");
            /* read in header info */
    bytes_read = fread( header_buffer, sizeof(ufix8),
                        sizeof(header_buffer), fp );
    if ( bytes_read == 0 ) {
        fclose(fp);
        fp = NULL;
        return("***** Error on reading font file header");
    }
            /* make it in terms of ORU's */
    ORUs_per_em = (ufix16) read_2b(header_buffer + FH_ORUPM);
    /* read the minumum buffer size for reading in characters */
    minimum_buffer_size = (ufix32) read_4b(header_buffer+FH_FBFSZ);
    if ( font_buffer != NULL )
        free( font_buffer );
    if ((font_buffer=(ufix8 *)malloc((ufix16)minimum_buffer_size)))
                == NULL) {
        fclose(fp);
        fp = NULL;
        return("***** Error: Unable to allocate memory for font buffer");
    }
    fseek( fp, (ufix32) 0, 0);      /* now load the fonts */
    if ( (bytes_read = fread((ufix8 *)font_buffer, sizeof(ufix8),
         (ufix16) minimum_buffer_size, fp) ) != 0 ) {
        fclose(fp);
        fp = NULL;
        return("***** Error on reading the fonts from the file");
    }
        /* now allocate minimum character buffer */
    minimum_char_buffer_size = read_2b(font_buffer+FH_CBFSZ);
    if ( char_buffer != NULL )
        free( char_buffer );
    if ((char_buffer =(ufix8*)malloc(minimum_char_buffer_size)) ==NULL) {
        fclose(fp);
        fp = NULL;
        return("***** Error:Unable to allocate memory character buffer");
    }
    font.org = font_buffer;
    font.no_bytes = bytes_read;
    first_char_index = read_2b(font_buffer + FH_FCHRF);
            /* Initialization */
    sp_reset(); /* Reset Speedo character generator */
            /* Set specifications for character to be generated */
```

continued

FONT WARS AND WHAT'S IN STORE

In September 1989, at the Seybold Conference on Computer Publishing, Apple and Microsoft announced they would both be using Apple's "Royal" font technology—Apple for its System 7 and Microsoft for Presentation Manager and Windows.

Royal (which has since been renamed TrueType) is an outline font technology designed for both screen and printer output. Apple worked with a number of font vendors in order to give Royal an extensive set of hinting algorithms. In addition, the Royal font format will be published—the intention obviously being to create a de facto standard by the sheer mass of the predicted market. If every Mac and every PM or Windows machine were able to make use of the Royal fonts, with the font software built into the operating system of these machines, it could turn out to be a considerable amount of market pressure indeed. The ability to take a document that was created on a Mac and get true WYSIWYG reproduction of it on your PC is nothing to be ignored.

This could be considered a direct attack on Adobe, since the intent of the Apple-Microsoft announcement was to squash PostScript with Royal. It was also intended to squash everyone else's font format, but Adobe was the one with the most to lose. Adobe fired back by publishing the format of their Type One fonts. Type One fonts are the Adobe-supplied fonts that can use PostScript's built-in hinting technology. Adobe had previously been pretty severe about protecting their font technology—Type One fonts were originally encrypted to prevent anyone from using the hinting techniques. Adobe had gone so far as to brag that they would hire any programmer who could break the encryption scheme (which ended up taking about three months...). Adobe had come under attack by some big guns, and was forced to start being a little more competitive in order to survive. Even so, their about-face may have come too late to prevent some major damage.

At the Seybold conference, IBM remained silent on the subject of what direction it would be taking in the near future, stating only that Royal would be supported. A bit more recently, IBM indicated that they will use Adobe as the preferred format, while supporting TrueType. Remember that IBM and NeXt have an alliance, which means all of that Display PostScript technology needs to be protected. The upshot of this is that PostScript and TrueType will be slugging it out for dominance in font technology, with PostScript available today (but only fully available on the Mac or in PostScript printers), and TrueType probably becoming available sometime in mid-1991.

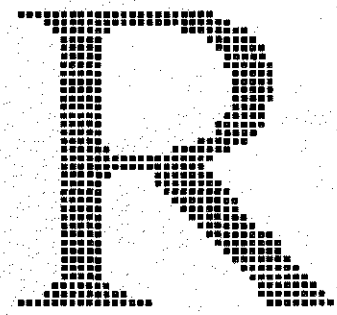
So the race is on, and now it's up to the contestants to show us their stuff. During the next few years, software and hardware manufacturers will have to come to a decision about which standard(s) to support. No matter what their final decision is, the biggest winners will be the users, who will be luxuriating in the thousands of fonts that will be available to us all.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 3: Normal and obliqued "R"s

A normal "R" is scaled to 12 points on a 300 dpi device. The formula for **xxmult** and **yymult** would be:

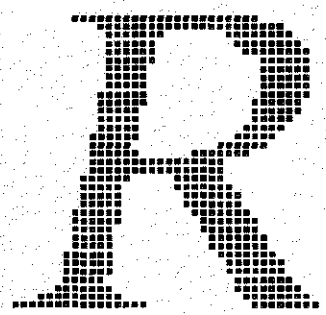
xxmult = 65536 * (12*300)/72,
  or 65536 * 50 pixels per em
yymult = 65536 * (12*300)/72,
  or 65536 * 50 pixels per em



Courtesy of Bitstream Inc.

An "R" is scaled to 50 pixels per em and obliqued 12 degrees clockwise (+12). The parameter **xymult** is set to 65536 * 50 * Tangent(12). The formula for **xymult** would be:

xymult = 65536 * 50 * 0.2125558,
  or 65536 * 10.6278



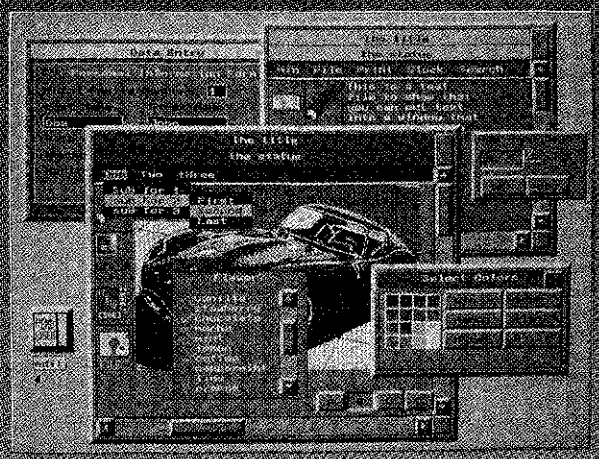
Courtesy of Bitstream Inc.

or querying the particular output device to find the physical resolution. The **main** routine takes care of finding this out, as well as of finding out other information related to the device—in this case, a VGA display. It then loops through a series of values of point sizes and *xymult* and *yxmult* combinations, all global variables, displaying a few lines of text and then waiting for a keypress.

After a set of font values has been specified, the routine **start_speedo** is called to begin the font generation process. It expects to be passed the name of the font file to be opened. The header information can be read from the font file once it has been opened. The header

NEW Microsoft graphics

graphics-MENU 3.0



A sophisticated graphical user interface for Turbo Pascal, Turbo C, Turbo C++ & Microsoft-C.

Automatic use of EMS (expanded memory) if it is available to save the underlying image.

MENUS: several types including horizontal & vertical nested to any depth, supporting keyboard and mouse. Auto save of underlying image.

ICONS: beveled or not, can be used as buttons or as application holders. Can be automatically aligned within a window or isolated.

BUTTONS: automatic arrays allow easy placement in or out of a window. Multiple modes allow use as radio buttons or check boxes.

WINDOWS: optional beveled look. Automatic drag and resizing. Vertical and/or horizontal scroll bars. Title and status lines - easily changed.

FORMS: true data-entry forms available with add-on module. Data validation, range-checking, pop-up menus, help lines, dBASE-like picture strings.

CASE TOOLS: on-screen design tools for code generation of menu structures and data-entry forms.

SOURCE: library source is available in all languages.

TASKING: Fore & background tasking capability allows GUI tasks to proceed as a background activity.

BGI, MS GRAPHICS or META: Works with Borland Graphics Interface, Microsoft graphics or MetaWindow from Metagraphics Software Corp.

NO RISK: 30-day money back warranty, free technical support, no licensing, no royalties.

$179 graphics-MENU
$379 graphics-MENU with source
$ 99 Data-Entry
$224 Data-Entry with source
$549 Pro-Pak (full src & lib's)

Specify BGI, META or Microsoft Graphics
ASK about support for Genius GX Graphics
Languages: Turbo Pascal 5.5, TurboC 2.0, TurboC++, MicrosoftC 5.1 (META only), or MicrosoftC 6.0 (META or MS graphics).

Island Systems

7 Mountain Road
Burlington MA 01803
VOICE (617) 273-0421
FAX (617) 933-1152

DEMO AVAILABLE

VISA and MASTERCARD accepted

Circle 317 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 4: Rotated characters

An "R" scaled to 50 pixels per em and rotated counterclockwise -90 degrees, the standard landscape orientation.

xxmult and **yymult** for normal characters (neither rotated nor obliqued) return values of 65536 multiplied by 50 pixels per em, as shown previously. The value for **rotation** is negative 90. The formula would be:

```
xxmult = 65536 * 50 * cos(-90),
  or 65536 * 50 * 0, or 0
xymult = 65536 * 50 * sin(-90),
  or 65536 * 50 * -1, or 65536 * -50
yxmult = 65536 * 50 * -sin(-90),
  or 65536 * 50 * -1, or 65536 * 50
yymult = 65536 * 50 * cos(-90),
  or 65536 * 50 * 0, or 0
```

Xpxls is now a function of negative **y**: **Youtl** determines **Xpxls** because the image has been rotated. Also, an **xoffset** has been provided to shift the rotated image vertically so all coordinates are positive.

Courtesy of Bitstream Inc.

provides information about the font, such as its name, the number of file characters, the minimum buffer size for character generation, etc. The **sp_set_specs** function is passed a pointer to the font buffer as part of a larger structure containing information about the size of the characters and the resolution of the target device. Once this information is passed to **sp_set_specs**, the font we want has been completely specified. We are then free to call **sp_make_char** to generate the characters. If the font needs to be changed in size or rotation, all that needs to be done is to reset the parameters and recall **sp_set_specs**. As the library calls out to **sp_set_bitmap_bits**, FONTDEMO uses the parameters passed to it to generate the raster lines of the character on the screen.

SUMMARY

Using this method, all the work of generating a new font's characters is done by the library, except for drawing the bitmaps and tracking the current position for the next bitmap. The benefit of adding a scalable font package to your program is being able to display a font in any size and in any position. As you can see, acquiring this ability requires a relatively small amount of code and delivers a big payoff. PJ

ProGraphx

ProGraphx Menus™

Simply add four lines of code to your program and you can have a pull-down or pull-up menu system. Define a menubar with a text file, compile it and receive events calls back to your program. Change the menubar without recompiling your program to create demo versions or other adaptations. A sample application of a Microcoded processor is included along with its two editors.

ProGraphx Menus™	$99
Menus with Source	199

ProGraphx Toolbox™

Over 50 hi-speed graphics routines for EGA/VGA 16 color modes, including proprietary icon routines to display color images.

ProGraphx Image Editor, a complete color icon paint package to edit and view up to 50 icons at once. Simply paint an icon, then use the ***ProGraphx Toolbox™*** routines to display it with amazing speed. Over **500** sample icons for use in your own programs. ***Operation:Airlift***, a full screen scrolling arcade style game with source code.

All routines written in assembly for fastest possible performance.

ProGraphx ToolBox™	$199
Toolbox with Source	399

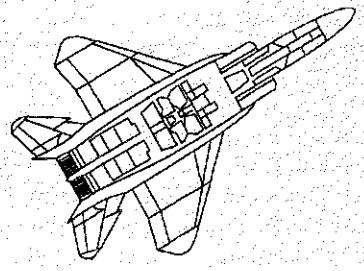
Stanwood Associates, Inc.
303 East Ohio Street Chicago, IL 60611

1-(800) 752-7982

(312) 828-9734 **VISA/MC** **FAX (312) 828-9742**

ProGraphx
Speed, Simplicity, and Power for Graphics Programmers

Compatible with Turbo C or Pascal, Zortech C/C++, Microsoft C or any language compatible with 80x86 object code. Uses less than 10k of your code segment!

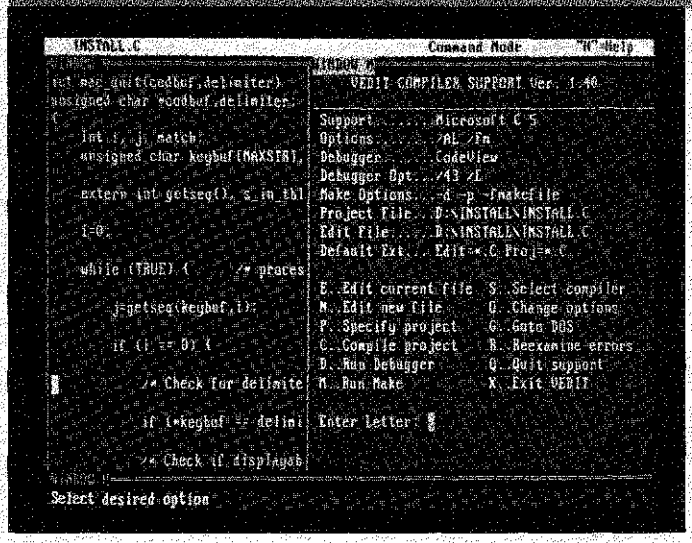


Also available from :
Programmer's Shop & Programmer's Connection

Circle **330** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, a fast, powerful text editor that integrates your favorite programming tools and uses no memory!



VEDIT PLUS

FREE
Fully Functional
Evaluation Copy
Call 1-800-45-VEDIT

- **Mouse Support**
- **Pull-down menus**
- **Columnar blocks**
- **1000 Level Undo**
- **Regular expressions**
- **Small 70K size, fast**
- **Emulate Wordstar, Word Perfect, Brief, vi, others**
- **Edit text and binary files of any size and line length**
- **Powerful macro programming language for instant "off-the-cuff" macros**

DOS XENIX UNIX QNX FlexOS

The new VEDIT family of text editors offers stunning performance, versatility and ease of use. Completely written in assembly language, they are small and lightning fast. (3 to 30 times faster than other editors on large files where speed really counts.) Edit text and binary files of any size, even 100+ megabytes. Installation is trivial; VEDIT.EXE and an optional help file are all you need - no overlays, no configuration files, no environment variables.

For programmers, the new compiler support in VEDIT and VEDIT PLUS is a breakthrough. Run not only popular compilers, but debuggers and your favorite tools from within the editor. When shelling to DOS, VEDIT swaps itself and any desired TSRs out of memory to give you more memory than when you entered VEDIT. Only VEDIT offers you the advantages of a powerful editor without giving up the convenience od an integrated environment.

Call for your free, fully functional, evaluation copy today. See why VEDIT has been the choice of 100,000 programmers, writers and engineers since 1980.

VEDIT Jr. - Unmatched performance for only $29.

All VEDIT editors include a pull-down menu system with "hot keys", context sensitive help, pop-up status and ASCII table, a configurable keyboard layout and flexible, unlimited keystroke macros. Perform block operations by character, line, file or column. Undo up to 1000 keystrokes - keystroke by keystroke, line by line, or deletion by deletion. Automatic indent, block indent and parentheses matching speed program development. Word wrap, paragraph formatting, justification, centering, adjustable margins and printing for word processing. Run DOS programs.

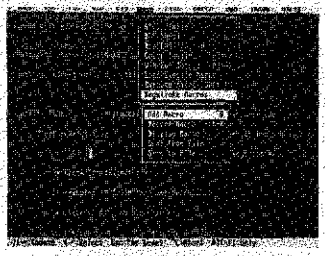
VEDIT - A best value at only $69.

VEDIT can simultaneously edit up to 36 files and split the screen into windows. Search/replace with regular expressions. The most integrated compiler support available. Run VEDIT PLUS macros.

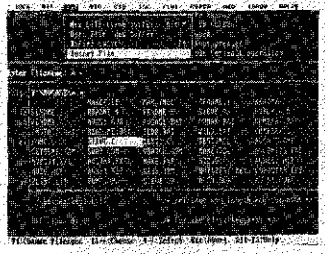
VEDIT PLUS - *Ultimate programmer's tool for only $185.*

VEDIT PLUS adds the most powerful macro programming language of any editor. It eliminates repetitive editing tasks and permits creating your own editing functions. The macro language includes testing, branching, looping, user prompts, keyboard input, string and numeric variables, complete control over windows plus access to hardware interrupts, memory and I/O ports. Source level macro debugging with breakpoints and tracing.

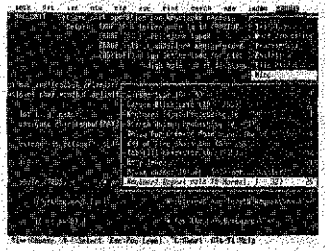
VEDIT PLUS - $185 for DOS, $285 for UNIX/XENIX. 30 Day Money-back guarantee. Discount pricing for multiple users, schools and OEMs. VEDIT Jr. site licenses start at only $250.



An intuitive user interface with pull down menus, hot keys, mouse support and context sensitive help make VEDIT easy to use, easy to learn.



Point and shoot file selection makes it a snap to edit new files, merge and split files, run macros and much more. Select files with the cursor keys or the mouse.



Just about everything about VEDIT is configurable, from the keyboard layout, to the screen colors, to the way control characters, tabs and the end of lines are displayed. Configure VEDIT with easy to use menus.

BENCHMARKS IN 3 MEG FILE

	VEDIT	BRIEF	Sage	QEdit
Simple search	57 sec	1:17 min	28 sec	Cannot
Save and continue	52 sec	3:52 min	1:47 min	Cannot
Load, modify, save, exit	21 sec	49 sec	1:38 min	Cannot
Block-column copy (40x200)	2 sec	30sec	2 sec	2 sec
Delete one column in file	**9:58 min**	**1:50 hour**	**1:03 hour**	Cannot
60,000 replacements	**3:18 min**	**1:44 hour**	**1:32 hour**	Cannot

VEDIT is a registered trademark of Greenview Data, Inc. BRIEF is a trademark of UnderWare, Inc. Sage Profession Editor is a trademark of Sage Software Inc. QEdit is a trademark of SemWare.

* DOS version also supports Concurrent DOS, DESQview, Microsoft Windows, PC-MOS/386 and most networks. Special CRT terminal version runs over serial lines and modems.

* Free evaluation disk is fully functional and can edit small files.

Greenview Data
P.O. Box 1586, Ann Arbor, MI 48106
(313) 996-1299, Fax (313) 996-1308

Circle **56** on Reader Service Card

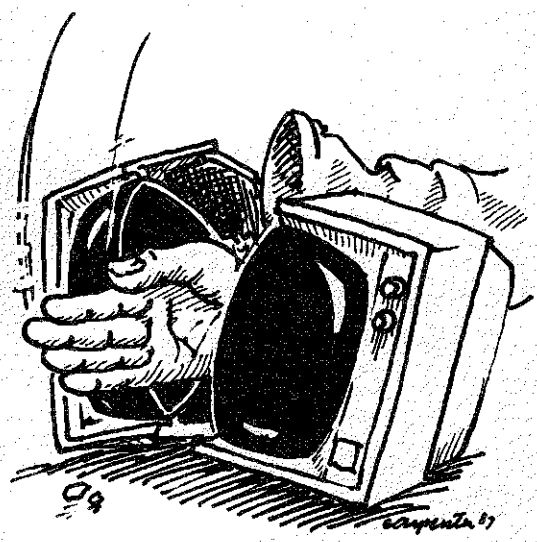
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    specs.pfont = &font;      /* Pointer to Speedo outline structure */
    specs.xoffset = 0L << 16;              /* Position of X origin */
    specs.yoffset = 0L << 16;              /* Position of Y origin */
    specs.flags = MODE_0;                  /* Mode flags */
    specs.out_info = NULL;
         /* the values are multiplied by 65,536 (shifted left 16)
            to conform to the internal representation used by Speedo.
            This retains high accuracy while using integer math only. */
    specs.xxmult = ( (unsigned long) (Point_x*PixelsperInch_x)<< 16 )
                 / (unsigned long) 72;      /* Calculate X */
    specs.yymult = ( (unsigned long) (Point_y*PixelsperInch_y) <<16 )
                 / (unsigned long) 72;      /* Calculate Y */
         /* Coeff of Y to calculate X pixels */
    specs.xymult = (unsigned long) (Value_xy)<< 16;
         /* Coeff of X to calculate Y pixels */
    specs.yxmult = (unsigned long) (Value_yx)<< 16;
         /* OK, now call into the library with these settings */
    if ( !sp_set_specs(&specs) )  {
        fclose(fp);
        fp = NULL;
        return("***** Cannot set requested character specs");
    }
        /* all went well */
        /* print out the values of the parameters at top of screen */
    settextposition(1,1);
    _settextcolor(6);
    sprintf(text_string,"%s X point=%d Y Point=%d  xyvalue=%u yxvalue=%u",
     (char *)(header_buffer+FH_FNTNM),Point_x, Point_y,Value_xy,Value_yx);
    _outtext( text_string );
    return( NULL );
}
/*** Now the routines required to use Speedo begin ***/
/*  sp_set_bitmap_bits:
 *  Three arguments, the horizontal raster line, followed by the
 *  beginning and ending vertical positions for a line to be drawn.
 *  This routine actually draws the character. */
void sp_set_bitmap_bits( fix15 y, fix15 x1, fix15 x2 )
{   POINT start, stop;
        /* convert from speedo bits to screen pixels */
    start.x = line_origin.x + char_origin.x + x1;
    stop.x  = line_origin.x + char_origin.x + x2 - 1;
    start.y = stop.y = line_origin.y - char_origin.y + y;
    _moveto( start.x, start.y );
    _lineto( stop.x, stop.y );
}
/*  sp_open_bitmap:
 *  The first call to start a character. Save the set_widths in order to
 *  move the current char position when done. The origins are where we
 *  initially offset the char from the starting location (some leading
 *  space). The size parameters are dimensions of the bitmap. Use only
 *  the vertical since the origin system is in the upper left corner. */
void sp_open_bitmap( fix31 x_set_width, fix31 y_set_width,
          fix31 x_org, fix31 y_org, fix15 x_size, fix15 y_size )
{
        /* save the x and y set widths to know where the next char starts */
        /* use integer math to retain accuracy */
    char_shift.x = (int)( (x_set_width+32768)/(65536) );
    char_shift.y = (int)( (y_set_width+32768)/(65536) );
        /* add in the leading spacing for this bitmap */
    char_origin.x += (int)( (x_org+(fix31) 32768) / (fix31) 65536);
    char_origin.y = y_size + (int)( (y_org+(fix31) 32768) / (fix31)65536 );
}
/*  sp_close_bitmap:
 *  Last call for a character. Use set_widths to move the current point
 *  to end of the character just drawn, in preparation for the next.     */
void sp_close_bitmap( void )
{       /* done with this character, so update the current char origin */
    char_origin.x += char_shift.x;
    char_origin.y += char_shift.y;
}
/*  sp_report_error: The library calls this when an error occurs. */
void sp_report_error( fix15 error_number )
{  /* SPEEDO errors generated during calls to library routines.*/
    static char *message[ ] = {
        "Insufficient font data loaded",
        "?",
        "Transformation matrix out of range",
        "Font format error",
        "Requested specs not compatible with selected output module",
        "?",
        "Intelligent scaling requested but not supported",
        "Unsupported output module requested",
        "Extended font loaded but only compact font supported",
        "Font specs not set prior to use",
        "?",
        "Character not available",
        "Track kerning request rejected for lack of required data in font",
        "Pair kerning request rejected for lack of required data in font",
    };
    RESET_VIDEO_MODE;                          /* reset video mode */
    printf("%s\n",message[error_number-1]);    /* display error message */
    exit(2);                                   /* quit the program */
}
/*  sp_load_char_data:
 *  Provides dynamic character loading to conserve memory. Dynamic loading
 *  isn't necessary if you read in the entire font file at once, but most
 *  people will trade speed for memory. Reads the requested number of
 *  bytes (no_bytes) from the file at an offset (file_offset) into the
 *  location in character buffer (cb_offset). Returns pointer to buffer. */
buff_t *sp_load_char_data( fix31 file_offset,
        fix15 no_bytes, fix15 cb_offset)
{
  /* Buffer descriptor for character data must be static
     since we are passing back a pointer to it */
    static buff_t char_data;
    int bytes_read;
    if ( fp == NULL ) {
        RESET_VIDEO_MODE;
        printf( "SP_LOAD_CHAR_DATA ERROR file never opened");
        exit(1);
    }
    if ( fseek( fp, (long) file_offset, (int) 0 ) != 0) {
        RESET_VIDEO_MODE;
        printf( "SP_LOAD_CHAR_DATA ERROR in seeking character");
        fclose( fp );
        fp = NULL;
        exit( 1 );
    }
    if ( (no_bytes + cb_offset) > minimum_char_buffer_size ) {
        RESET_VIDEO_MODE;
        printf( "SP_LOAD_CHAR_DATA ERROR Character buffer overflow");
        fclose( fp );
        fp = NULL;
        exit( 1 );
    }
    bytes_read = fread( (char_buffer+cb_offset),sizeof(ufix8),no_bytes,fp);
    if ( bytes_read != no_bytes ) {
        RESET_VIDEO_MODE;
        printf( "SP_LOAD_CHAR_DATA ERROR on reading character data");
        fclose( fp );
        fp = NULL;
        exit( 1 );
    }
    char_data.org = (ufix8 FONTFAR *)char_buffer + cb_offset;
    char_data.no_bytes = no_bytes;
    return( &char_data );
}

/* functions patterned from some in Bitstream's nsample.c program that
 * reads one, two, and four bytes from the font buffer */
static ufix8 read_1b( ufix8 *pointer )       /* read 1 byte  */
{   return *pointer;
}

static fix15 read_2b( ufix8 *pointer )       /* read 2 bytes */
{   fix31 temp;
    temp = *pointer++;
    temp = (temp << 8) + *(pointer);
    return( (fix15) temp );
}

static fix31 read_4b( ufix8 *pointer )       /* read 4 bytes */
{   fix31 temp;
    temp = *pointer++;
    temp = (temp << 8) + *(pointer++);
    temp = (temp << 8) + *(pointer++);
    temp = (temp << 8) + *(pointer  );
    return( temp );
}
```

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 80 and 90.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Windowing for two



A Windows 3.0 DLL to access a second monochrome monitor

By Chris Schwartz

Life is full of irony. Just as real estate values are plummeting nationwide, screen real estate is becoming an increasingly valuable asset for people migrating to GUIs such as Windows. Even with a big bad NEC 5D 20-inch monitor and the Windows-requisite monochrome monitor for CodeView, I'm still cramped for screen space. This DLL provides the solution.

After many late-night sessions with CodeView (which is actually usable this time), I decided to figure out how to gain access to my second monitor. I brushed up on the memory management chapter in the *Guide to Programming*, and thought this project would be a piece of cake. The KERNEL.EXE program exports psuedo-selector values for common real mode segments, including B000h for the monochrome monitor. However, these values turned out to be useless—they did nothing but cause general protection faults (another misadventure with the SDK, as it turned out).

DPMI TO THE RESCUE

After a little reflection, I stumbled onto the DOS Protected Mode Interface (DPMI) specification that I had ordered a month earlier, which is the key to this puzzle. The scoop on DPMI is that Intel, Microsoft, Lotus, and other software and hardware manufacturers got together and solved the communication issue between protected mode programs and DOS services such as TSRs and device drivers. DPMI is a complicated beast, and since magazine real estate is as valuable as screen space, I'll refer you to Ray Duncan's excellent article in the July 90 issue of *Microsoft Systems Journal*. The actual specification is free from Intel at 800-548-4725 (ask for part number JP-26).

Instead of digging into DPMI, I will demonstrate how to use what is probably the most important DPMI function for Windows programmers: accessing Real Mode Memory from Standard and Enhanced Modes. Let me note that this certainly has wider applications than simply writing on the monochrome monitor.

Couple writing to the screen with the ability to generate DOS software interrupts and to communicate with products like Btrieve and any other custom resident DOS services, and you'll see how the ability to work with DPMI is worth the price of admission.

USING MONOSCRN.DLL

Now on to MONOSCRN.DLL. This module consists of only four exported functions: two for DPMI and two for Vio Services. Since **VioInit** demonstrates how to use both DPMI services, let's begin there. **DpmActiveMode** determines the mode (real or protected) the code is executing in. It is provided so we can decide whether or not we ought to call the function **DpmSelectorFromSegment**. This function, oddly enough, does exactly what it says.

Let's take a jargon break here. The proper term for the CS, DS, ES, and SS registers during real mode execution is *segment*; in protected mode they are called *selectors*. The distinction is made because in real mode they point to physical memory and in protected mode they are really index values into a physical memory table, typically the Local Descriptor Table (LDT).

NOW FOR THE VIDEO INTERFACE

In keeping with the concept of data hiding (in which DLLs are treated as "black boxes"), a function named **VioInit** should be called first. **VioInit** calls **DpmActiveMode** to determine the execution mode. If executing in protected mode, we ask for a selector for the monochrome segment by using **DpmSelectorFromSegment**; if we're not in protected mode, **DpmActiveMode** sets *wVideoSeg* to B000h.

The last function is modeled after the OS/2 Vio interface. **VioWrtStrAttr** displays a buffer of bytes with a specific attribute byte. I created an internal **ScreenOffset** function to accommodate

---

Chris Schwartz is president of Softworks, which specializes in Windows and OS/2 software development. Contact Chris at (404) 876-6115.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; Module:         Monochrome Screen access from Windows 3.0
; Description:    This module demonstrates how to access memory mapped
;                 devices within the Windows environment.
; Author:         Chris Schwartz  (c) 10/25/90

                .MODEL SMALL
                PUBLIC  DllInit         ;dll entry point
                PUBLIC  WEP             ;dll exit point
                PUBLIC  VioInit, VioWrtStrAttr, DpmActiveMode

                PUBLIC  DpmSelectorFromSegment

                .DATA
bogusData       dw      8 DUP(0)        ;to keep windows from wigging out
wVideoSeg       dw      0

                .CODE
;       BOOL DllInit(VOID); - Dll entry, called by windows during loading
;       Entry Values:  DS = Library Data Segment    ES = Command Line Selector
;                      DI = Module Handle           SI = Command Line Offset
;                      CX = Local Heap Size
;       Return Values: AX  = 0 Failed, don't load
;                      AX != 0 Success, load this puppy.
DllInit         PROC    FAR
                mov     ax, 1           ;load this puppy
                ret
DllInit         ENDP

;       FAR PASCAL Calling Convention
;       BOOL WEP(WORD wExitMode); - Dll exit point
;       Return Values: AX  = 0 Failed Cleanup
;                      AX != 0 Success
WEP             PROC    FAR
                mov     ax, 1           ;goodbye
                ret     2
WEP             ENDP

;       BOOL DpmActiveMode(VOID);
;       Return Values: AX  = 0 Protected Mode under DPMI
;                      AX != 0 Real mode, or not under DPMI
DpmActiveMode   PROC    FAR
                mov     ax, 1686h       ;DPMI Mode Detection Op Code
                int     2Fh             ;only changes the AX register
                ret
DpmActiveMode   ENDP

;       FAR PASCAL Calling Convention
;       WORD DpmSelectorFromSegment(WORD)
;       Return Values: AX = 0 Then function failed, else we have a selector.
;------- Formal Parameters ------------------
wRealSegment    equ     [bp+06]
DpmSelectorFromSegment  PROC    FAR
                push    bp
                mov     bp, sp
                mov     ax, 02
                mov     bx, wRealSegment
                int     31h
                jnc     DpmSFSExit      ;return protected mode selector
                xor     ax, ax          ;return error code
DpmSFSExit:     pop     bp
                ret     2
DpmSelectorFromSegment  ENDP

;       VOID VioInit(VOID);
VioInit         PROC    FAR
                push    ds
                mov     ax, DGROUP
                mov     ds, ax          ;fixup dll's data segment
                call    DpmActiveMode
                or      ax, ax
                je      VioInitProtMode
                mov     wVideoSeg, 0B000h
                ret
VioInitProtMode:
                mov     ax, 0B000h      ;monochrome segment value
                push    ax
                call    DpmSelectorFromSegment
                or      ax, ax
                je      VioInitExit
                mov     wVideoSeg, ax
VioInitExit:
                pop     ds
                ret
VioInit         ENDP

ScreenOffset    PROC    NEAR
                mov     bx, ax
                mov     cx, 6
                shl     ax, cl
                sub     cx, 2
                shl     bx, cl
                add     ax, bx
                add     ax, dx
                shl     ax, 1
                mov     dx, wVideoSeg
                ret
ScreenOffset ENDP

;       FAR PASCAL Calling Convention
;       VioWrtStrAttr(lpData, wLen, wRow, wCol, lpAttr)
;------- Formal Parameters ------------------
lpData          equ     [bp+16]
wLen            equ     [bp+14]
wRow            equ     [bp+12]
wCol            equ     [bp+10]
lpAttr          equ     [bp+06]

VioWrtStrAttr   PROC    FAR
                push    bp
                mov     bp, sp
                push    ds
                mov     ax, DGROUP
                mov     ds, ax          ;fixup dll's data segment
                push    es
                push    di
                push    si
                mov     ax, wRow
                mov     dx, wCol
                call    ScreenOffset  ;returns proper video address
                or      dx, dx
                je      VioWrtStrAttrExit
                mov     es, dx
                mov     di, ax
                mov     cx, wLen
                lds     si, lpAttr
                lodsb
                mov     ah, al
                lds     si, lpData
VioWrtStrAttrLoop:
                lodsb
                stosw
                loop    VioWrtStrAttrLoop   ;cook those puppies
VioWrtStrAttrExit:
                pop     si
                pop     di
                pop     es
                pop     ds
                pop     bp
                ret     14
VioWrtStrAttr   ENDP
                END     DllInit
```

LISTING 2

```
;   Monoscrn.def
LIBRARY   Monoscrn
EXETYPE   WINDOWS
CODE      PRELOAD MOVEABLE DISCARDABLE
DATA      PRELOAD SINGLE
EXPORTS
    WEP                     @1  RESIDENTNAME
    DpmActiveMode           @2
    DpmSelectorFromSegment  @3
    VioInit                 @4
    VioWrtStrAttr           @5
```

LISTING 3

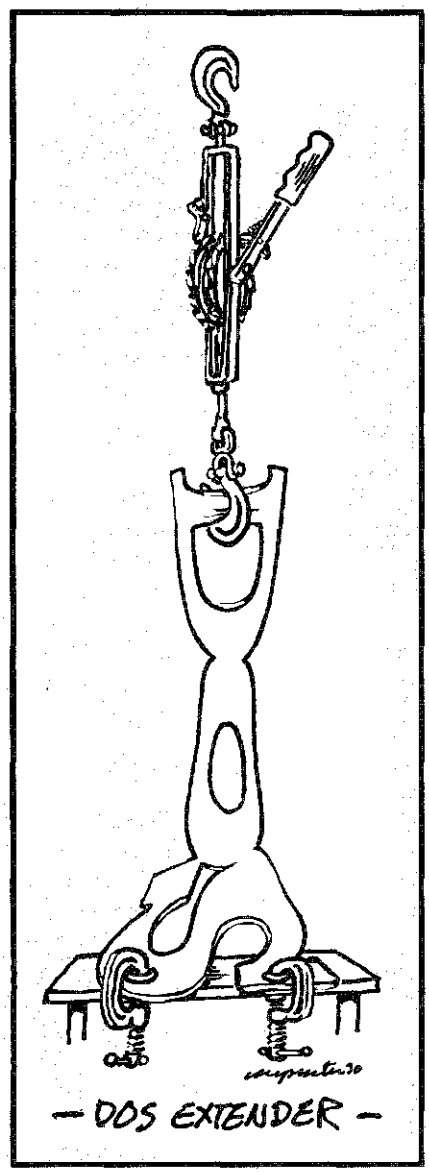
```
# Monoscrn.mak   10/25/90
monoscrn.obj:   monoscrn.asm
      masm /Zi monoscrn.asm;
monoscrn.dll:   monoscrn.obj monoscrn.def
      link /AL:16 /M /CO monoscrn,monoscrn.dll,monoscrn.map,,monoscrn.def
      mapsym monoscrn.map
      implib monoscrn.lib monoscrn.def
      rc monoscrn.dll
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the addition of more display functions. For all the mystery and intrigue that is associated with protected mode, this job was exceedingly easy. Ironically, real mode is the most dangerous time to access the monochrome monitor because Windows might be using B000h for an EMS frame.

DLL DESIGN AND WINDOWS

If we want to get sophisticated, we can move **VioInit** to the DLL entry procedure so we won't have to bother calling it. This is in line with the less-is-more DLL design school (i.e., the fewer entry points, the better). To really make **VioInit** a robust module we would need to introduce the concept of windows or active screen sessions.



This is necessary to protect against a situation where two separate programs ask for the monochrome selector. Under Windows, they will get the same selector and thus will be able to overwrite each other.

The reason for this deserves some attention. Windows 3.0 is very different from OS/2 and 386 versions of UNIX in that, instead of protecting the programs from one another, Windows sets each program up in the same address space (i.e., in only one LDT). This is what allowed me to overwrite CodeView's display when using this module as it now stands. So while Microsoft has been promoting Windows' uses of protected mode, the story is only half told. The decision to use a single LDT is not without consequences; it creates the very real opportunity for applications to step on each other and wreak havoc for everybody. Hmm ... if you're thinking this sounds like the same old problems you had with DOS and TSRs, you're absolutely right.

Beware protected mode under Windows—it's not what you might be led to believe. I find this situation particularly interesting since Microsoft took Novell to task over a similar design flaw in Netware 386. But hey, what goes around comes around.

WRAPPING IT UP

Let me wrap this thing up by pointing out the value of this DLL for non-professional programmers using ToolBook and other scripting languages that allow access to DLLs. Past experience has shown that client power users really eat up additional features like the ability to use two monitors because it makes them look smart. And Confucius says there's nothing like a happy client.

In a future issue, I'll explore the subject of Windows' Input Hooks and how to use them so we can distribute keystrokes and move the mouse pointer back and forth between two monitors. This, of course, is the missing piece to building a character-based GUI to run alongside Windows. I've got a client running a custom e-mail application on Novell using this setup who finds it indispensible. Just think of the custom workstation opportunities that are yet to be discovered.

Until next time, Happy Hacking! PJ

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk	$25 US	*Call for prices for other processors.*
Manual only	$75 US	
AMX 86	$3000 US	

(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 211 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GUI madness

Or why I still use character-based interfaces

By Brett Glass

There are times—more and more times, nowadays—when I swear I've had it with GUIs.

At the risk of dating myself, I'll reveal that I have been programming computers since the time of "stone knives and bear skins"—that is to say, punched cards. When I wrote my first program, I did it on a keypunch in FORTRAN; I then handed my stack of cards to an operator who fed it into the RJE terminal of a massive IBM mainframe. When my program was run by the end-user, he'd have to keypunch his data, add his deck of cards to mine, then submit the whole stack and wait for output. Slow? Yes. Inconvenient? Absolutely. We've come a long way since that time—personal computers are as fast as that old mainframe, and sport some bright, colorful screens and fancy graphical user interfaces.

As each new GUI came out, I tried it with great curiosity and gusto. (I have developed substantial programs for the Macintosh, GEM, Amiga, OS/2 Presentation Manager, Windows 3.0, two flavors of Smalltalk for the PC, and X Window, among others.) Yet the more GUI programming I do, the more I feel that the companies providing the computers and operating environments have not been as diligent in offering incentives to programmers as they have in hyping the not-always-substantial benefits of GUIs to end-users.

This is not to say that I'm entirely down on GUIs; I'm not. However, I do believe it's time for software developers to clamor for advances that help *them* produce code that's portable, fast, and efficient—and also easy to write, debug, maintain, and (most of all) use. In this article, I'll address some of these issues and discuss some ways in which current GUIs can be improved.

Brett Glass is a free-lance writer, programmer, and consultant residing in Palo Alto, California. He can be reached at rogue@well.sf.ca.us on Internet, on CompuServe at 72267,3673, or as GLASS on Bix.

STANDARDS AND PORTABILITY

One advantage of the old-fashioned Writeln (or printf, PRINT, etc.) statement is that it's pretty much guaranteed to work on any machine—even one with a GUI. This method of communication may not be terribly robust, but it's sometimes handy to be able to give the user a message without large amounts of code, time, and effort. If the language you're using is at all portable, you should be able to get your program running on just about any system.

Alas, this is not true of GUIs. One of my programming clients, for example, made a terrible mistake in the fall of 1989. He listened to the COMDEX announcement by IBM and Microsoft that developers ought to work on new programs for OS/2 first and for Windows second. A year later, he's in deep financial trouble and has given up a large share of his company to venture capitalists in a desperate bid for funding.

Why? Because despite IBM's and Microsoft's "commitment," OS/2 is selling poorly and Windows 3.0 has taken the market by storm. Microsoft has since announced that OS/2 2.0 will be able to run Windows applications, and the Windows to OS/2 Software Migration Kit (SMK) will allow Windows programs to be ported to OS/2 1.X with very little effort. In other words, the strategy that was advocated by the behemoths a year ago was exactly the wrong thing to do.

DEVELOPER BOONDOGGLE

My client is bitter; he accuses Microsoft of intentionally sending developers on a "boondoggle" so that it could get its applications out first. Machiavellian theories aside, his travails highlight a basic problem with current GUIs. Programs take a long time to port, generally requiring massive rewrites when they are moved from one environment to another. This is especially true if one uses features such as OS/2's "threads," which aren't present in most other environments. If the fundamental paradigms differ greatly—as they do, say, between the Mac and Windows—it may be easier to start from scratch.

A few environments have begun to emerge that run on many platforms. X11, developed as part of MIT's Project Athena, runs on most UNIX systems. (There are also implementations for the Mac under A/UX and the Amiga, and Quarterdeck has just announced a multitasking environment called DESQview/X for the IBM PC.) The problem here is that unless the interface comes with the computer, it becomes an extra-cost option, potentially raising the price of the software and introducing an unfamiliar user interface. Alas, as long as computer manufacturers continue to think there's something to be gained by remaining incompatible, developers—and ultimately users—will suffer.

The other approach to the problem is to create a development environment that hides the differences between GUIs from the user. One system I've had good luck with is ParcPlace's ObjectWorks,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

which lets you develop for UNIX, OS/2, and Windows. The problem here is that Smalltalk is not for everyone—especially developers with large bodies of code in other languages.

There's no panacea for these problems, but manufacturers can help by realizing that programmers *will* port their code if it's easy to do so. The minor advantages of a neat gimmick in a proprietary environment are dwarfed by the advantages of a standard API.

LANGUAGE SPECIFICITY

Another important problem with most GUIs now is that nearly all are designed and documented to work primarily with a single language. NextStep, for example, is strongly tied to Objective-C. Windows, OS/2 PM, and X11 are tough to use with anything but C. If you prefer to program in a type-safe, structured language like Modula-2, good luck—you'll have a hard time finding bindings or documentation.

If you do find them, odds are they will be developed by the compiler vendor—and thus they won't even be portable between compilers! (On my shelf I have three Modula-2 compilers for OS/2, each with completely different bindings for OS functions.) The Macintosh had a similar problem: the *Inside Macintosh* reference manual was written with the assumption that all programmers would use Lisa Pascal.

Now, there are a *lot* of programmers out there who have large investments in code bases in other languages, including FORTRAN, Pascal, BASIC, and COBOL (in which more lines of code are still written daily than in any other language). And it just so happens that I, as a programmer, am one of those folks who have doubts about the suitability of C for the creation of reliable, professionally-crafted software. Whatever reasons programmers have for choosing a different tool, they ought to be able to preserve their current code investment.

A SIMPLE ANSWER

Fortunately, the answer to this problem is simple. Manufacturers and compiler vendors must be willing to take the time to get together and standardize on an API for each GUI—not just in one language, but in all the major languages. The programmer documentation for the GUI should then be printed with as many bindings as possible. That way, a brilliant programmer who likes to work in, say, Prolog or LISP will have just as easy a time porting an application as the die-hard C programmer.

THE COMPLEXITY OF THINGS

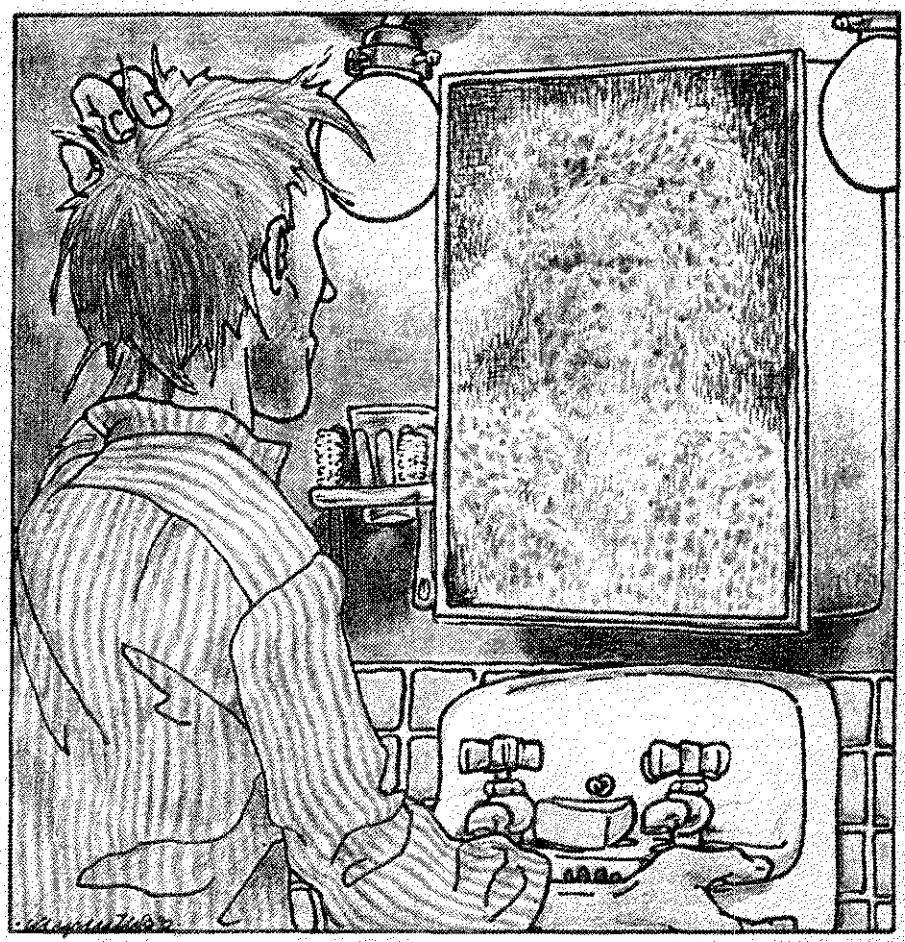
Anyone who wants to write Microsoft Windows applications will likely start by looking at GENERIC.C, the "generic" Windows program included with the SDK. It's a bit daunting, however, to discover that this minimal program (which does nothing but display an "About..." box) is several pages long, and it takes a day of study to fully understand everything in it!

OS/2 PM has roughly the same level of complexity, and the minimal Mac program is a bit shorter (more of the work is done automatically or hidden in the resources and Event Manager). In none of these environments can you write a graphical "Hello, world!" program in a few short lines—unless your compiler maker has implemented things for you in the runtime library.

There is no inherent reason why this should be so. In an earlier essay ("The Importance of Good Defaults," *PJ* 8.1), I stated that it should be possible to make most or all of the actions that one performs most often happen by default. In my wanderings among the GUIs, I've seen some rather clever and (alas) little-known ways to do this. For instance, in AmigaDOS, if one opens a "pseudo-file" with a special name that specifies the width, height, and position of a window, the window pops open and everything that's written to that file appears in the window.

RAW SPEED

It's expensive, both computationally and in terms of bandwidth, to draw an image pixel by pixel rather than by specifying a whole character to go into a particular slot on the screen. It may be worth it when doing purely graphical work. But is it worth slowing down all



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

output on the chance that graphics just might be useful?

Proponents of graphical user interfaces will argue that the average overhead of a GUI is small, because it's only incurred when the computer writes to the screen. But what this really means is that the delay is concentrated at the worst possible moment—when the user has made a request and is waiting for an answer! Unless the program absolutely requires graphics, many programmers will opt out of using a GUI if raw speed is a consideration.

Even when using graphics, a programmer may be tempted to avoid using the GUI and go directly to the hardware for maximum speed. The answer here is the same as for text: provide hardware support for text and graphics sufficient to eliminate the speed penalty. If writing text to a GUI's screen is as simple and fast as writing to the screen buffer of a character-based display card, there will be no reason NOT to use the GUI for everything. And the shorter the path between the application and the hardware, the faster everything will be displayed.

THE MACRO PROBLEM

One problem I feel has never been addressed adequately in GUIs is that of macros. This isn't because designers haven't been thinking about the problem; rather, it's because there's no *a priori* way for the environment to discern the intentions behind certain actions, such as mouse movements, so that it can mimic them correctly.

Let's look at a simple example to see why. Suppose the macro recorder sees a user move the mouse halfway across the screen, stop, reverse direction, and click on an icon representing the file FOO.DAT. What information should the recorder save for next time? Should it remember the exact path the mouse took and replicate it on the screen when the macro is invoked? (This could be problematic if the macro is next run on a machine with a screen that has a different resolution or aspect ratio, or if the windows are in different positions.)

Should it look for a file with the same name? (It might not exist!) What if the user picks a menu item during recording, and it's disabled during playback? There's no way—unless the computer is psychic—to know what to do in these situations. Un-graphical as it might be, the only solution is a text-based macro language (which few GUIs have) that avoids this sort of ambiguity. The GUI should implement this language rather than requiring the programmer to create a new (and possibly unique) scheme for the application.

NO MORE TRAINING WHEELS

One aspect of GUIs that can be infuriating is the lack of built-in ways to "take off the training wheels" for advanced users. A primary purpose of GUIs is to make a program or system easier to use for the naive or inexperienced user. This is great for the system or software vendor whose bottom line is affected primarily by that initial sale. But in the long run, the user may pay for this short-lived benefit many times over with slow, awkward operations for which there are no good shortcuts.

Most GUIs do have "shortcut" keys for items on menus, and these are indeed handy. However, there are many operations that don't appear on the menus, and so can't have shortcut keys. Since GUIs operate with a paradigm that's based on picking things manually rather than describing them, the ability to use "wildcards" is often absent. Any process that doesn't have a good graphical analog may be difficult or impossible to perform using a GUI. In these situations, there is no good option other than a command language with substantial expressive power. If the GUI doesn't provide it, the programmer should.

Being forced to use a mouse to size and position objects may make precise operations difficult. Some programs, such as Xerox Ventura Publisher, get around this problem by allowing you to specify the exact location of an object via a dialog box. (This lets you make a postcard which is precisely 4.25 inches long without squinting at a "ruler" on the screen.) Many applications, however, don't offer this feature, and most need it. For this reason, it would be useful to have precision positioning implemented as part of the GUI itself.

CAN YOU AFFORD A GUI?

With Microsoft Windows 3.0 selling in many places for $99 or less, the question of GUI affordability does not seem—at first glance—to be much of an issue. It's only when you get the environment out of the box and try to *use* it that you realize where the real investment needs to be made: in a fast CPU, lots of RAM, a good monitor and graphics card, and a nice, 100+ MB hard disk.

The Macintosh GUI once carried a similar cost penalty built into the price of every machine. The situation may have improved this fall with Apple's introduction of cost-reduced Macs. However, you still may need to add to the relatively small amounts of RAM initially supplied with these systems to get good performance.

In the Intel world, at least, there is now a way to avoid much of the memory overhead of the GUI without being incompatible with it. Many developers have chosen to use DOS extenders, rather than Windows, as the basis for new programs whether they're graphical or not. A DOS extender program that is equipped to work with the DOS Protected Mode Interface (DPMI)—such as the version of Rational's DOS/16M which Lotus used to produce its latest version of 1-2-3—can run in protected mode on just about any machine.

In fact, it will work not only under Windows, but also under multitasking environments (such as DESQview) that support VCPI, UNIX "compatibility boxes," and ordinary DOS. This may well be the answer to universal compatibility on Intel-based machines.

A TIME OF TRANSITION

It's important to keep all of these potential pitfalls and problems in perspective. GUIs are still a relatively new technology, and will doubtless get better over time. The important thing is for user interface developers not to rest on their laurels or to ignore one another's innovations (or to attempt to prevent healthy competition by litigation).

Ultimately, no matter how good GUIs get, there will be times when a simple, character-based interface is not only enough for the job at hand but a highly desirable and practical solution. For instance, I'm writing this article on a clear, crisp, 50x80 character-based VGA screen that lets me see a large portion of the article at once. Had I started with a GUI, I might also have gotten the job done—but I bet I'd still be working on it now.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tear Your C Development Time in Half.

We understand the pressure you're under, trying to create, test, and debug complex C programs without Instant-C™, the professional development environment. With all your searching for errors and recompiling, you barely have time to test code, never mind read magazines.

Instant-C can make your time twice as productive. Here's how:

Industrial-strength debugging. Instant-C finds bugs other compilers miss, automatically detecting over 500 errors, including:

- *Array references out of bounds*
- *Stack overflow*
- *Use of uninitialized pointers*
- *Overwriting a return address*
- *Read or write through null pointers*
- *Overwriting your program or DOS*

Ultra-fast, incremental compiling. When you make a change to a program, Instant-C recompiles just the change. In fact, Instant-C can rebuild a 100,000 line program in less time than it takes to read our headline. And in recent tests, Instant-C was 24 times faster than Microsoft Programmer's WorkBench.*

Microsoft® C compatibility. Instant-C links with Microsoft C libraries to form an enhanced development environment. After developing, testing, and debugging with Instant-C, you can use Microsoft C to make the final version of your program.

Unlimited program size. Instant-C uses Rational Systems' DOS/16M technology, the software that enabled Lotus to break the 640K barrier for 1-2-3® Release 3. Because it uses extended memory, no program is too big, and your work isn't slowed down by swapping to disk.

That's not the half of it. Instant-C is a complete development environment, including source-level debugging, dynamic cross referencing, lint-like static check, support for third-party libraries, and interactive testing of single functions or partial programs.

The Instant-C Environment

- The industry's fastest edit-compile-link-debug cycle
- Configurable editor and source formatter
- Incremental compiler and linker
- Direct access to 16MB of memory
- Extensive static and run-time error checking
- Multiple virtual or real screens
- Source-level debugging
- Support of commercial libraries
- Conditional breakpoints in C
- Compatibility with assembly language object modules
- Immediate evaluation of C expressions, including macros
- Compatibility with Microsoft C compiler 5.1 and 6.0
- Interactive execution of functions
- Allows use of Microsoft run-time libraries
- Dynamic cross-referencing and browsing
- Full graphics screen support

Get the whole story. To find out more about how Instant-C can double your productivity, give us a call. We'll give you all the details in a flash. After all, we know how valuable your time is.

Rational Systems, Inc.

Rational Systems, Inc.
220 N. Main St.
Natick, MA 01760
Toll-free: (800) 666-2585
Outside U.S.: (508) 653-6006
Fax: (508) 655-2753

* Tests involved changing one line of code in a 11,500-line program, running on a 20MHz 80386 system with 8MB memory. Instant-C requires an 80286/386/486 and at least 1M of memory. Instant-C is a trademark of Rational Systems, Inc. Microsoft is a registered trademark of Microsoft Corp. 1-2-3 is a registered trademark of Lotus Development Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Text buffers in windows

Start of a character-based windowing system in Turbo C/C++

By Allen Holub

Though Turbo C—and Microsoft C, for that matter—comes with a nice set of graphics and text-manipulation primitives, there's no support for real windows. I suppose Microsoft leaves window functions out deliberately as motivation to purchase MS Windows. Turbo C's **window** function is a step in the right direction—it lets you define an area of the screen that can be addressed in a window-like manner, but it's not full window support. Text that's written to the window doesn't wrap correctly, there's no automatic scrolling when the window fills, and so on.

WINDOWING—TAKE ONE

This article is the first of two parts that present a reasonable windowing system for Turbo C (though the port to Microsoft C isn't particularly difficult). The system supports various video adapters, allowing 43- and 50-line screens on the EGA and VGA. It supports line wrap and scrolling in windows, boxed windows and various other window operations, as well as basic editing features (insert characters and lines, delete characters and lines, etc.).

LOOK THROUGH ANY VIEWPORT

The most important feature, however, is that the underlying text buffer (which I'll call the "window") can be much larger than the aperture into that text buffer which is displayed on the screen (the "viewport"). The text buffer can be made to slide either horizontally or vertically beneath the viewport. This feature is particularly useful in data-entry screens—and that's what windows are all about. It lets you do things like have a 40-column-wide field in a data-entry screen which is actually a viewport into a 128-column window.

When the text that you type exceeds the 40 columns on the screen, the text starts sliding under the viewport as you add characters, until you reach the window's full width. The effect is similar to an editor's horizontal-scrolling feature. You can also access multiple-line windows through a single-line viewport, using the cursor to bring hidden lines into view.

As usual, I've built a complex system out of several simple layers which communicate with one another over a well-defined function interface. That is, no layer ever directly manipulates another layer's data. This object-oriented approach has two advantages: first, any layer can be replaced with relative ease without affecting the other layers, provided that the function-calling interface remains unchanged. Second, since that function-calling interface has to be designed carefully, the code tends to be quite usable in the general case—in applications outside the current one. A side benefit to this structure is that it's easy to reuse the same code in C++ classes. This way, one body of subroutines can be used in both C and C++ programs, because only the interface mechanics change.

The code this month is the text-manipulation component of the window system, collectively called the *textlib*. Most of these functions aren't functions—they are small enough to merit being macros, and you get the added benefit of faster execution. A handful of real functions is also included, each in its own source-code file, so that the final program won't be made unnecessarily large by calling in unused functions from the library.

For present purposes, a "text buffer" is conceptually a two-dimensional array of characters in which each row represents a line. The buffer is encapsulated in a **textbuf** structure, along with various other fields that are needed for buffer manipulation. In the current implementation the buffer is really a two-dimensional array, though you could change this to a data structure which is

Allen Holub is a teacher, writer, consultant and composer. He teaches C and compiler design for the U.C. Berkeley Extension. His most recent book is Compiler Design in C. *Reach him electronically at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

more conservative of memory (such as an **argv**-like array of string pointers).

A significant amount of space can be lost in the current implementation because of the large number of blanks that are likely to be at the right edge of the page. On the positive side, a single two-dimensional array is easier to manipulate than more efficient data structures, so the code that does the manipulation is going to be relatively speedy.

Because of the space problem, I have tried to use memory flexibly and efficiently. The actual buffer can be stored in either near or far memory. This way, a small-model program can put its buffers in the far heap—outside of the data area of the program entirely—so that the windows take up very little of the available 64K. On the other hand, since far-heap access can be less efficient than near-heap, it's easy to change things back to near-heap storage (see Listing 1).

HEAP SELECTION

The code that is executed when the **#ifdef NEAR_HEAP** at the top of Listing 1 is active sets up memory for near-heap operation. *BMALLOC* and *BFREE* map to **malloc** and **free** function calls, and the maximum buffer size (*MAXBSIZE*) is limited to 4096 bytes. A pointer to one character in the text-buffer (**bufptr**) is defined as a simple character pointer and the **bufsize** type, which is guaranteed to hold the buffer size, is mapped to **int**. The far heap is used when **NEAR_HEAP** isn't defined. Here, *BMALLOC* and *BFREE* map to **farmalloc** and **farfree**. (With Microsoft C, use **_fmalloc** and **_ffree** .)

The buffer size is now limited to 64K-less-one elements. (The -1 is required by ANSI C, which says that a pointer must be able to go one character past the end of an array. This restriction effectively takes one element away from the 64K maximum imposed by far pointers.) A **bufptr** is now **typedef**ed to a **far*** rather than to a simple pointer, and the buffer size (*bufsize*) is now mapped to an **unsigned int** to take care of the extra available space.

The final heap-related issue has to do with buffer initialization and buffer copying. Normal **memset** and **memmove** calls can be used for near-heap buffers, but special functions (**_bufset** and **_bufmove**) must be provided to manipulate far-heap buffers. The **bufset** and **bufmove** macros in Listing 1 map to one or the other of these functions as appropriate. The functions themselves are declared in Listings 1 and 2.

THE TEXTBUF STRUCTURE

TEXTBUF.H continues with the definition for a **textbuf**, which looks like this:

```
typedef struct
{
    unsigned magic;
    int nrows;
    int ncols;
    int row;
    int col;
    bufptr p;
    bufptr buf;
}
textbuf;
```

The *magic* field holds an arbitrary number which the different functions use to validate the structure. When a **textbuf** is allocated, the field is initialized to the symbolic constant *B_MAGIC* and set to 0 when it's freed. The *nrows* and *ncols* fields are the array's dimensions (number of rows, number of columns), and *row* and *col* indicate the current cursor position within the array. Finally, *buf* is a pointer to the actual array, and *p* points at *buf*[row][col].

TEXTBUF INITIALIZATION

Following the **textbuf** definition in Listing 1 are prototypes for four real functions used to allocate and initialize text buffers. The first set of functions is used like **fopen** and **fclose**, as in the following example:

```
f()
{
   textbuf *p;
   p = b_new(nrows, ncols);
    ...
   b_delete( p );
}
```

The **b_new** function is passed the buffer size in rows and columns. It allocates and initializes the buffer and returns a pointer that's used by the other buffer-manipulation functions. Delete the buffer with a **b_delete** call when you're finished with it.

An alternate allocation strategy is provided for those situations where you want to allocate the **textbuf** structure yourself, like this:

```
textbuf buf;
b_construct(&buf,nrows,ncols);
b_destruct( &buf );
```

or like this:

```
struct fred
{   ...
    textbuf b;
} *fredp;
fredp = malloc( sizeof(*fredp) );
b_construct(&fredp->b,nrows,ncols);
b_destruct( &fredp->b );
free(fredp)
```

This second set of functions is particularly useful in the foregoing example—when an actual **textbuf** (as compared to a **textbuf** pointer) is encapsulated in another structure. C++ programmers will also note that these functions can be used as constructors and destructors with minor modifications. Both **b_new** and **b_construct** return a **textbuf** pointer on success and **NULL** on failure.

BEWARE OF SIDE EFFECTS

The functions themselves are declared in Listings 1 through 4. All of the rest of the text-manipulation routines are handled by macros in TEXTBUF.H (see Listing 1). Most of these macros have side effects, so be careful. The arguments with side effects are summarized in Table 1. Note that a C++ version of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
/* textbuf.h */
#ifndef __TEXTBUF_H     /* Make sure that textbuf.h isn't included */
#define __TEXTBUF_H     /* twice, the endif is on the last line.   */

#ifdef NEAR_HEAP
#   define BMALLOC   malloc      /* Allocate memory for buffer.  */
#   define BFREE     free        /* Free memory used for buffer. */
#   define MAXBSIZE  4096        /* Maximum row * col size.      */
    typedef char     *bufptr;   /* pointer to the buffer        */
    typedef int       bufsize;  /* holds max buffer size        */

#   define bufset( dst,c, size) memset (dst,  c,  size)
#   define bufmove(dst,src,size) memmove(dst, src, size)
#else
#   include <alloc.h>           /* needed for farmalloc prototype */
#   define BMALLOC   farmalloc
#   define BFREE     farfree
#   define MAXBSIZE  (0xffffU - 1) /* 64K - 1 */
    typedef char far* bufptr;
    typedef unsigned  bufsize;

    extern void _bufset  ( bufptr p, const int c, bufsize size );
    extern void _bufmove ( bufptr dst, bufptr src, bufsize size );
#   define bufset(p,c,s)        _bufset(p,c,s)  /* in textbuf.c */
#   define bufmove(d,s,size)    _bufmove(d,s,size)
#endif

#define PUBLIC  /* for documentation: globally accessible object. */

typedef struct
{
    unsigned magic; /* Magic number for verification    */
    int nrows;      /* Number of rows                   */
    int ncols;      /* Number of columns                */
    int row;        /* Current cursor position, row     */
    int col;        /* Current cursor position, column  */
    bufptr p;       /* Character at that position       */
    bufptr buf;     /* Actual buffer.                   */
}
textbuf;

/* B_LAST_LINE is passed a textbuf pointer and returns a pointer to
 * the beginning of the last line of the buffer. The first line is
 * pointed to by "buf."
 *
 * textbuf.magic is set to B_MAGIC when a textbuf is allocated. Use
 * it to verify that a textbuf pointer is reasonable.
 */
#define B_LAST_LINE(b)  ((b)->buf + (((b)->nrows - 1) * (b)->ncols))
#define B_MAGIC 0xabcdU
/*-------------------------------------------------------------*/
PUBLIC textbuf *b_new( int nrows, int ncols );
PUBLIC void     b_delete( textbuf *b );

PUBLIC textbuf *b_construct( textbuf *b, int nrows, int ncols );
PUBLIC int      b_destruct( textbuf *b );
/*-------------------------------------------------------------*/
#define b_current( b )  ( (b)->p        )
#define b_advance( b )  ( (b)->col++,   \
                          (b)->p++      \
                        )
#define b_row(b)        ( (b)->row      )
#define b_col(b)        ( (b)->col      )
#define b_nrows(b)      ( (b)->nrows    )
#define b_ncols(b)      ( (b)->ncols    )
#define b_setrc(b,r,c)  ( ((b)->row = (r)),                     \
                          ((b)->col = (c)),                     \
                          ((b)->p   = (b)->buf + ((r)*(b)->ncols)+(c)) \
                        )
#define b_ateol(b)      ( (b)->col == ((b)->ncols - 1) )
#define b_ateob(b)      ( (b)->row == ((b)->nrows - 1) )
#define b_valid(b,r,c)  ( (b)                                   \
                          && ((b)->magic == B_MAGIC)            \
                          && (0 <= (r) && (r) < (b)->nrows)     \
                          && (0 <= (c) && (c) < (b)->ncols)     \
                        )
#define b_clearbuf(b,c) ( b_setrc(b,0,0),                       \
                          bufset((b)->buf, (c), (b)->nrows * (b)->ncols)\
                        )
#define b_cr(b)         ( ((b)->p -= (b)->col ),        \
                          ((b)->col = 0       )         \
                        )
#define b_lf(b)         (  ++(b)->row           ),      \
                           ( (b)->p += (b)->ncols )     \
                        )
#define b_clearline(b,c) (  b_cr(b),                            \
                            bufset( (b)->p, (c), (b)->ncols ) \
                         )
#define b_available(b)  ( (b)->ncols - (b)->col )
#define b_insert_c(b,n,c) ( bufmove( (b)->p + n, (b)->p,                \
                                     b_available(b) - (n) ), \
                            bufset ( (b)->p    , (c)   , (n) )  \
                          )
#define b_delete_c(b,n,c) ( bufmove( (b)->p, (b)->p+(n), (n) ),         \
                            bufset ( ( (b)->p - (b)->col) +             \
                                     ( (b)->ncols - (n) ), (c), (n))    \
                          )
#define b_closeup(b,c)  ( b_cr(b),                              \
                          bufmove( (b)->buf + (b)->ncols, (b)->buf,     \
                                   (b)->row * (b)->ncols ),     \
                          bufset( (b)->buf, (c), (b)->ncols )           \
                        )
#define b_closedown(b,c)  ( b_cr(b),                                    \
                            bufmove( (b)->p, (b)->p + (b)->ncols,       \
                                     ((b)->nrows-(b)->row-1) * (b)->ncols),\
                            bufset( B_LAST_LINE(b), (c), (b)->ncols )   \
                          )
#define b_openup(b, c)  ( b_cr(b),                              \
                          bufmove( (b)->buf, (b)->buf + (b)->ncols,     \
                                   (b)->row * (b)->ncols ),     \
                          bufset( (b)->p, (c), (b)->ncols )             \
                        )
#define b_opendown(b,c) ( b_cr(b),                              \
                          bufmove( (b)->p + (b)->ncols, (b)->p,         \
                                   ((b)->nrows-(b)->row-1)*(b)->ncols),\
                          bufset( (b)->p, (c), (b)->ncols )             \
                        )
#endif /* #ifndef __TEXTBUF_H */
```

LISTING 2

```
/* _bufset.c */
#include <stdlib.h>
#include "textbuf.h"

/* You can't use the standard memset() and memmove() functions for
 * buffers because they may have been allocated from the far heap.
 * The bufset() and bufmove() calls are mapped to memset() and
 * memmove() when NEAR_HEAP is true, otherwise they are mapped to
 * calls to the following, equivalent, far-pointer functions. Bufset
 * is made a function, even though it's short enough for a macro,
 * to avoid side effects on "size" and "p."
 */

void _bufset( bufptr p, const int c, bufsize size )
{
    while( size-- > 0 )         /* Assume size can be unsigned */
        *p++ = c;
}

void _bufmove( bufptr dst, bufptr src, bufsize size )
{
    if( dst < src )
        while( size-- > 0 )
            *dst++ = *src++ ;
    else
    {
        dst += size;
        src += size;
        while( size-- > 0 )
            *--dst = *--src;
    }
}
```

LISTING 3

```
/* b_construct.c */
#include <stdlib.h>
#include "textbuf.h"

PUBLIC textbuf *b_construct( textbuf *b, int nrows, int ncols )
{
    /* Initialize a textbuf. Return b on success, NULL on failure */

    bufsize asize = nrows * ncols;

    if( asize >= MAXBSIZE )         /* Won't fit in an 8086 segment */
        b = NULL;
    else if( !(b->buf = BMALLOC(asize)) )  /* get the actual buffer */
        b = NULL;
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

NEW!	GUIDO (Graphical User Interface with Dialog Objects, multiple hot regions per window, data entry functions, window minimization to icon)	$400
NEW!	M++ (fast C++ math classes; indexing, matrices & vectors, numerics, memory handling, I/O)	$395
Windows!	C++/Views (C++ interface to MS Windows 3.0; over 60 classes; free broswer (no source); no royalties; Zortech C++ only)	$395
NEW!	MBALLOC (best-fit far heap manager, replaces standard routines, Microsoft performance on Turbo C)	$340
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
NEW!	ZIP Image Processing Library (brightness, contrast, merge images, TIFF/GIF/PCX/bin, printer & HRT 512 digitizer support)	$290
	Victor Image Processing Library (EGA/VGA; TIFF/PCX/GIF/bin; multiple images; many filters; frame grabber & live video)	$250
	TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Install 3.05 (automatic installation program; user-selected partial installation; CRC checking)	$195
	Wendin-DOS Plus (self-bootable, multitasking, multiuser MS-DOS replacement; includes XTC editor)	$180
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	Rogue Wave tools.h++ Class Library (collections, B-trees, file space management, time& date, persistent objects; complete user's guide)	$165
	Rogue Wave math.h++ Class Library (C++ overloadings for standard operators, matrix inversion & FFT; complete user's guide)	$165
NEW!	PCX Effects (fades, wipes, animation, music)	$165
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$165
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	Hold Everything (spawn new programs; swap parent to EMS or disk; handles video, interrupts, & environment; returns error level)	$150
Windows!	RED (program development editor for Microsoft Windows; keystroke recording & playback; unlimited undo)	$150
	C Generator (generates C code to read & write file records defined with C structure syntax)	$150
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
	C Communications Toolkit by Magna Carta (multi-port & co-processor board support, FAX, interrupt driven, emulations, xfer protocols)	$125
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	Envoy Communications Package (includes editor, script language, redefinable keys, help screens, menu-driven interface; Turbo C)	$120
Windows!	C++ Class Library for Windows 3.0 (over 100 classes, requires Win3.0 SDK & Glockenspeil or Zortech C++)	$120
	Kier CountryLib (date, time, measurement, numeric, and simple language conversion; helps build country-independent software)	$120
	SilverWare "C" EMM Library (60 interface functions to EMM Version 4.0)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$115
	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, Clarkson drivers, Stan's NFS clone, Bdale mailer)	$100
	Disassembler (8086/80286/80386/80C186 with 80287/80387/80C187 NPX)	$90
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
NEW!	NIH Class Library including hardback version of the book *Data Abstraction and Object-Oriented Programming in C++*	$80
Windows!	Cview Windows 3.0 Program Editor (multiple fonts, C formatting, SDK help, auto install)	$80
Windows!	WinMem (unlimited global memory handles for Windows 3.0, 4-byte overheadper block rather than 20, virtual memory for 286)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	COMM-DRV (robust serial communications driver; COM1-COM25; INT 14H & EBIOS; FOSSIL & multi-tasker compatible; inc. file transfer)	$75
NEW!	Software Tools for C (145 functions: TSR, EMS, interrupt handling, screen functions with screen generator, etc.)	$70
	XSPAWN (swap parent process to expanded memory or disk before running child)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
NEW!	MultiStr (evaluates a string containing multiple ASCII arithmetic expressions with variables)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
NEW!	Network Toolbox/N (C functions for Novell Netware 2.1x, sample programs included)	$55
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
NEW!	CALC (ASCII algebraic expression evaluator, unlimited parenthesis nesting, symbols, 32 built-in functions, easily extended)	$50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	Window Boss & Data Clerk (windows, menus, data checking, on-line help, window drag, screen forms)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
NEW!	YATE (Yet Another Text Editor; written in C++, full-screen)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available at additional cost)	$50
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Classy Forms (screen forms in both C and C++ versions)	$40
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	Spell Pack (6 spelling programs & a hyphenator: Ispell, Microsp, Sp, Cspella, Spell, Hy, & the Dawg utilities)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	6-Pack of Editors (baker's half-dozen public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
Upgrade!	Crunch Pack (25 file compression & expansion programs)	$30
	CCALC (handy extended-precision calculator; real and complex models; many built-in functions)	$30
NEW!	String Pack (lots of string routines; C++ String class, BAWK, word wrap, fuzzy search, Boyer-Moore, Hypertext, etc.)	$25
	PC-MAIL (UUCP mailer by Wietse Z. Venema; send, receive, and manage UUCP mail)	$25
	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language; includes hardcopy docs)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; BSD Version 2.3 with docs)	$25
	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
	VAL Linker by David Troendle (a complete linker; excellent for learning about linking)	$20
Cheaper!	OOPS Class Library (collection of basic C++ classes by Keith Gorlen; Version 3.0 (book); uses C++ R2.0)	$20
	Data	
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$80
	Dictionary Word List (234,932 words in alphabetical order)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	each $30
	Interactive Computer Ephemerii (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.15 (C source))	$30

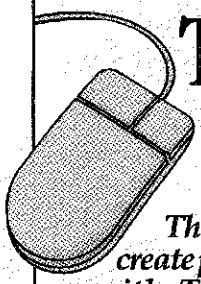
The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3464 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

UNLEASH TURBO SPEED



The fastest way to create powerful programs with Turbo Pascal 5.5, Turbo C 2.0 and Turbo C++!

POWER TOOLS PLUS™/5.1 – $149

– lightning-fast routines to help you:

◆ Add easy-to-use integrated mouse support for windows and menus ◆ Generate context sensitive help screens ◆ Resize and move windows and use drop shadows for that professional look ◆ Let users choose from window-oriented pick lists ◆ Create and access "huge" data structures ◆ Use multiple-line edit fields with fully configurable edit keys ◆ Add EMS support ◆ Write TSRs and ISRs easily ◆ Create powerful programs in Turbo Pascal 4.0, 5.0 & 5.5!

Turbo C TOOLS™/2.0 – $149

– fast, high quality functions to help you:

◆ Add easy-to-use integrated mouse support for windows and menus ◆ Quickly include virtual windows and menus ◆ Integrate your windows and menus with Turbo C's text windows ◆ Create context-sensitive help screens ◆ Provide multiple-line edit fields with fully configurable edit keys ◆ Write TSRs and ISRs easily ◆ Create powerful programs in Turbo C 1.0, 1.5, 2.0 and Turbo C++!

FREE with these products!

All source code, complete sample programs, and a comprehensive manual are included. We offer free technical support and a bulletin board dedicated to technical issues.

Unleash your potential!

We offer programming tools that are fast, flexible and affordable. Call now to order, or to ask for a free brochure on our full line of products for C and Pascal.

Put Blaise tools to the test!

If during the first 30 days you are not satisfied, we'll refund your money.

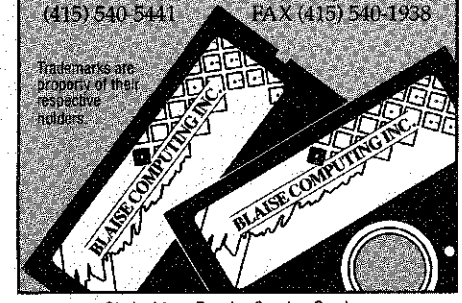
Call (800) 333-8087 today!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710

(415) 540-5441 FAX (415) 540-1938

Trademarks are property of their respective holders.



Circle 14 on Reader Service Card

these macros could implement them as inline functions, thereby eliminating the side-effect problem.

I use the comma (sequence) operator quite heavily here to put together multiple-statement macros. The comma operator evaluates left to right and evaluates to the right-most thing in the list. Taking **b_advance** as characteristic, the macro is defined as follows:

```
#define b_advance( b )  \
        (  (b)->col++,  \
           (b)->p++     \
        )
```

This macro accesses the current cell in the array and advances the pointer one notch. If *p* is a pointer into a standard two-dimensional array, this macro works like *p*++. The following invocations:

```
*b_advance(b) = x;
y = *b_advance(b);
```

are equivalent to these operations:

```
*p++ = x;
x = *p++;
```

The macro first increments the *col* field of the **textbuf** pointed to by *b*, and it then advances the current-position pointer. Because the comma operator evaluates to the right-most thing in the list, expressions involving **b_advance** work just as if the **(b)-> col++** weren't there (at least for purposes of assignment). Note that this macro is typical in that it's meant to be used as a buffer primitive, so there's no test for wrap to the next line when the column position gets too large.

Other macros are provided to do this adjustment as necessary, but the testing isn't built into **b_advance** because it would slow the macro down when testing wasn't required. Dealing with word wrap is left to the next layer up, which will use the **textbuf** functions to do any required testing.

CURSOR MANIPULATION

There are a number of other **textbuf**-manipulation macros in TEXTBUF.H. (Note that in the following discussion, *b* is a **textbuf** pointer, *r* a row index, and *c* a column index.) The first batch of macros lets you access and manipulate the actual characters. The macro **b_current**, which is related to **b_advance**, allows you to modify or access the current buffer element without advancing the position pointer. Use **b_current** in the following manner:

```
*b_current(b) = 'x';
i = *b_current(b);
```

The **b_setrc** macro lets you change the cursor position to an arbitrary row and column. As in **b_advance**, the macro doesn't check for a valid row and column—it just goes where you tell it. The first of the following examples sends the cursor to a specified row and column. The second line does that, but also puts an "x" at the new position. The third line fetches the character at the new position.

```
b_setrc(b,r,c);
*b_setrc(b,r,c) = 'x';
i = *b_setrc(b,r,c);
```

All of the foregoing functions actually write a character to the buffer. This means that an attempt to write an **'\n'** will just put a funny IBM graphics symbol on the screen. The macros **b_cr** and **b_lf** let you do the equivalent of a carriage return and a line feed, respectively. Note that these macros work like the raw ASCII characters, not like C functions. A **b_cr** call puts the cursor over the left-most character on the current line; **b_lf** moves you down to the same column on the next line—you aren't positioned at the left edge of the next line unless you issue both commands.

TABLE 1: Side effects of macros

Macro Name	Side effects on these arguments
b_current(b)	none
b_advance(b)	b
b_row(b)	none
b_col(b)	none
b_nrows(b)	none
b_ncols(b)	none
b_setrc(b,r,c)	b r c
b_ateol(b)	b
b_ateob(b)	b
b_valid(b,r,c)	b r c
b_clearbuf(b,c)	b
b_cr(b)	b
b_lf(b)	b
b_clearline(b,c)	b
b_available(b)	b
b_insert_c(b,n,c)	b n c
b_delete_c(b,n,c)	b n c
b_closeup(b,c)	b
b_closedown(b,c)	b
b_openup(b, c)	b
b_opendown(b,c)	b

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    else
    {
        b->nrows = nrows;
        b->ncols = ncols;
        b->row   = 0;
        b->col   = 0;
        b->p     = b->buf;
        b_clearbuf( b, ' ' );
        b->magic = B_MAGIC;
    }
    return b;
}
```

LISTING 4

```
/* b_destruct.c */
#include <stdlib.h>
#include "textbuf.h"

PUBLIC b_destruct( textbuf *b )
{
    if( b->buf && b->magic == B_MAGIC )
    {
        BFREE( b->buf );
        b->magic = 0;
        return 1;
    }
    return 0;
}
```

LISTING 5

```
/* b_new.c */
#include <stdlib.h>
#include "textbuf.h"

PUBLIC textbuf *b_new( int nrows, int ncols )
{
    textbuf *b;

    if( b = malloc( sizeof(textbuf)) )
        if( !b_construct( b, nrows, ncols ) )
        {
            free( b );
            b = NULL;
        }

    return b;
}
```

LISTING 6

```
/* b_delete.c */
#include <stdlib.h>
#include "textbuf.h"

PUBLIC void b_delete( textbuf *b )    /* Free a textbuf allocated by   */
{                                     /* a previous b_malloc().        */
    if( b_destruct( b ) )             /* Explicitly declared textbufs  */
        free( b );                    /* can be destroyed with an ex-  */
}                                     /* plicit b_destruct() call.     */
```

LISTING 7

```
/* texttest.c */
#include <stdlib.h>
#include <stdio.h>
#include "textbuf.h"

#define NROWS 3         /* Size of test buffer */
#define NCOLS 8
/*-------------------------------------------------------------------*/
void printbuf( textbuf *b, int verify )
{
    /* Print the buffer. The cursor position is indicated by
     * parenthesizing the buffer entry at the current cursor
     * position. If "verify" is true, an error message is
     * printed if the buffer does not hold its original
     * contents (as set up in main()):  a b c d e f g h
     *                                  i j k l m n o p
     *                                  q r s t u v w x
     */

    int i, row, col, startrow, startcol, expected;

    startrow = b_row(b);
    startcol = b_col(b);

    expected = 0;
    for( row = 0; row < NROWS; ++row )
```

continued

NEW V 4.1

Q PRO 4 TM

lean, mean, programming machine

WORLD'S FASTEST! MOST COMPACT!

4GL FOR APPLICATIONS DEVELOPMENT PROS

Q-PRO 4 is the best 4GL for real professional applications developers. This compact, professional quality, rock solid interpreter is field proven with over **25,000** installed. For DOS/LANs/ PCMOS-386 with full record and file lock – Novell and NetBios! ConDOS version native mode (.CMD files & code sharing – up to 4 users in 640k!).

Blazing File and Screen Speed!

Q-PRO 4's speed rivals compiled C. Runs nicely on a 4.77 mhz XT and flies on an AT! For example:

6 min. – Write 10,000 unique 255 byte records and update 4 indices. Using this file:

24 secs. – Add 100 records at random locations.

0.2 secs. – Position to a record by its primary index; display the record's 50 fields formatted on the screen.

Amazingly Compact

The entire Q-PRO 4 interpreter is less than 50k bytes of raw power! Your whole application will need less than 300k of RAM. You won't need memory stretchers or other DOS workaround gimmicks!

Its complete 115 command syntax and advanced 4GL event driven architecture speed your development. Easily does 1-to-1, 1-to-many and many-to-many relationships. Extensible — add your own routines in C or assembler! Optional distribution system encrypts and locks your code for safe distribution. No royalties!

Q-PRO 4 includes: unique multiple table driven data entry screens with full color control; dialog boxes; light-bar menus; selection windows; data dictionary; report Generator; query language (optional).Can your DBMS do this?

3 Commands – Retrieve a record using its key and display all its fields to the screen.

***2* Commands** – Write a full screen (up to 1900 fields) to an indexed file and update all keys.

NO Commands – Build a complete data entry screen with fill-in-the-blanks entry fields, prompts and field edits (justification, type checking, must tab, etc.).

Order Q-PRO 4 now! Half price introductory offer: $200.00 (+7.75 S&H) until 12/31/90. 30 day guarantee. Check, VISA, MC, COD. Demo/eval systems available.

QNE International®

136 Granite Hill Court Langhorne, PA 19047

TEL: (800) 333-0448 FAX/MODEM: (215) 968-3550

(All timings on 8mhz AT Clone, 30 ms disk.)

Circle **60** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Developing C++ Applications?
Introducing...
Linaeus
The C++ class browser and design tool
Graphical trees let you examine complex inheritance hierarchies, whether the classes are in one file or hundreds on multiple drives.
Instant access to detailed information about every class on your machine - just click on its box to view its data items, methods, constants, ancestors, descendents and filespec.
Valley queries let you find your class or subset quickly.
Interfaces with Brief.
Full GUI environment - DESQview required, mouse and EMS supported.
$195
Also...
VMO - Virtual Memory Objects
A dynamic memory structures class library. Objects can grow to huge gigabytes using heap, EMM and disk. Transparently. Do things like:
Memset(p[obj2,obj1,100000000L);
Strings, arrays, lists, matrices, trees, heaps more. Variable length element sizes to 4gb. Imagine a tree node that contains the text to The Lord of the Rings! Ideal for image manipulation and DBMSs.
Included is our standard class library with extensive input, file, control, messaging, debug, objects and simulated parameterized types.
Over 70 classes, 1000 methods.
$295
Zircel software
(303) 750-9543

As was the case with the previous macros, there's no check for running off the end of the buffer if you use too many **b_lf** invocations.

WHERE ARE WE NOW?

Several macros are provided for checking the current position in the buffer so that you can test for boundary conditions before issuing any of the movement requests. These macros evaluate to the current row and column position of the cursor [the upper-left corner of the buffer is (0,0)]:

```
b_row( b )
b_col( b )
```

The **b_available(b,n)** macro returns the number of characters to the right of the cursor on the current line (i.e., the distance from the cursor position to the right edge of the line). It's something like a reverse **b_col**.

Two macros test for end-of-line and end-of-buffer. The macro **b_ateol(b)** evaluates true if the cursor is positioned at the right edge of the line, and **b_ateob(b)** evaluates true if the cursor is resting on the bottom line.

Another testing alternative would be **b_valid(b, row, col)**, which evaluates true only if *b* points at a legitimate **textbuf** (one returned from **b_new** or initialized by **b_construct**), and if *row* and *col* form a legal cursor address.

Two more macros can occasionally be useful:

```
b_nrows( b )
b_ncols( b )
```

Both evaluate to the number of rows and columns in the indicated **textbuf**. The same numbers are passed to **b_new** or **b_construct**, but the macros save you the trouble of storing them yourself.

EDITING MACROS

The remainder of the **textbuf** macros give you editing capability: **b_clearbuf(b,c)** homes the cursor and fills the entire buffer with the character specified in the *c* argument.

The **b_clearline(b,c)** macro does the same thing with the current line: the cursor is placed at the left edge of the current line, and the entire line is then filled with the character specified in *c*.

The **b_insert_c(b,n,c)** macro inserts *n* c's at the current cursor position and moves everything located to the right of that position over a notch to make room. The cursor position is not modified, so it will end up under the left edge of the inserted space.

Characters at the far right of the line (the ones that are pushed off) are lost. No test is made to see if *n* characters are available on the line, so it's a good idea to call **b_available** before calling this function. The following is also safe:

```
n = min( b_available(b), n );
b_insert_c(b,n,' ');
```

The complement of **b_insert_c** is the macro **b_delete_c(b,n,c)**, which deletes *n* characters from the line, starting at the current cursor position. Everything to the right of the deleted characters on the line is moved to the left to fill in the space that has been created.

The character specified in *c* is used to fill in the far right side of the line. Again, it's up to you to check that the cursor is at least *n* characters from the right edge of the line, using something like this:

```
if( b_available(b) >= n )
   b_delete_c(b,n,' ');
```

DELETING LINES

The remaining macros all take care of adding and deleting entire lines from the buffer. The functions **b_closeup(b,c)** and **b_closedown(b,c)** both delete the current line. The **b_closeup** function moves all text above the current line down one line and fills in the new top line with the character specified in *c*.

As you might have guessed, **b_closedown** does the opposite: it moves all text below the current line up one line, then fills in the new bottom line with the specified character. The cursor ends up at the far left of the current line, which will now hold the text that was originally above or below the line, depending on which function you called. For example, if an eight-by-three buffer looked like the following:

```
a b c d e f g h
(i) j k l m n o p
q r s t u v w x
```

(the cursor is at the i, thus the parenthesis), then a call to **b_closeup(b,'@')** would yield:

```
@ @ @ @ @ @ @ @
(a) b c d e f g h
q r s t u v w x
```


Circle 75 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Programmers Know What Works

Get a Good Start

When you find something that works, you keep using it. When that something is versatile and reliable, you use it a lot. C is that kind of tool. *SuperFunctions* is that kind of library.

Expanded Memory. Pull-Down Menus. Mouse for text and graphics. A Critical Error Handler. These are things you'll need. Maybe you'll be writing a package to do program trading on the Stock Exchange. Maybe your program must decode satellite data on atmospheric ozone.

Whatever it does, every program needs some basics. Like memory management. Like a user interface. You could write these functions yourself, but you don't need to, because they are already in *SuperFunctions*. Don't do what has already been done. Get *SuperFunctions* and get on with your real work!

Expanded Memory

Take along all the luggage you want. If your programs need more than 640k of memory, you can have it with *SuperFunctions*. Sense installed expanded memory. Use the **LIM 4.0** spec to map in pages and pages of expanded memory as data segments. Up to 32 megabytes. Effectively, **memory limitations will disappear** for you.

Critical Error Handler

Get some insurance. The days of letting DOS write "Abort, Retry, Ignore" right on top of your beautiful interface are gone. *SuperFunctions* gives you an elegant means to handle open drive doors, disk, and other critical errors. When a critical error occurs, your program can **intercept the error and recover** using the function in this library.

Logical Windows

Simply a necessity. Every beginner book on C has a chapter on text windows, and that's nice -- for beginners. You'll need more than that. *SuperFunctions* gives you more than 250 logical windows. You can move any one of them to the top with a single function call. You can delete a window and update the screen with a single function call. Write to a window which isn't even on the screen or one that is partially covered by another window. Keyboard idle functions keep the screen alive even when the user is not responding. Write to the window using lots of different text attributes, different colors, different intensities, word wrap on or off, etc. With *SuperFunctions*, it's all simple, automatic, worry free, and fast.

Menus

Use a good map. People use menus because they're simple, useful, and they work. *SuperFunctions* gives you menus that are **simple, useful, workable, and powerful**. One easy to use function determines how big the menu windows should be, displays them, deletes them when finished, accepts first-letter key matches, and indicates the selection to the calling program.

Mouse

Your programs need to know how to control a mouse. *SuperFunctions* gives you that power. **Mouse detection, click detection, position detection.** Complete interface for graphic and text modes.

Extras

SuperFunctions is loaded with extras. But we're running out of room here. We'll just mention that it has **CMOS Timer and Alarm, DOS file functions, advanced time and date, command line parsing,** and more.

Nuts and Bolts

Greenleaf's *SuperFunctions* is a nuts and bolts library with functions you'll really need. It will help you go a long way, and if necessary, you can actually get inside it and fine tune things yourself, because we give you all the source code. **FREE.** So have a ball.

FREE Bumper Sticker

Sure, a bumper sticker is silly. We know that. But it's also kind of fun, and when it's done right, so is programming. Find out more about our libraries and get a FREE bumper sticker, call us.

1-800-523-9830

Ask for extension #200. Also, don't forget to ask about our Gold Support Program.

FAX: (214) 248-7830

SuperFunctions™...$299.

GREENLEAF Software©

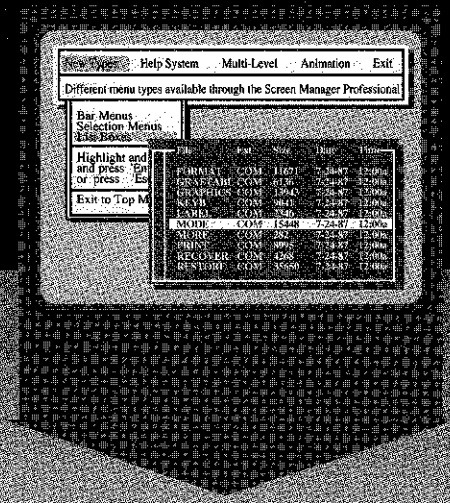
Greenleaf Software, Inc.
16479 Dallas Parkway #570
Dallas, TX 75248

Circle 21 on Reader Service Card.

PROPERTY OF DOS.ORGANIZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    {
        for( col = 0; col < NCOLS; ++col )
        {
            b_setrc(b, row, col);
            if( *b_current(b) == 'a' + expected++  ||  !verify )
            {
                if( startrow == row && startcol == col )
                    printf("(%c)", *b_current(b) );
                else
                    printf(" %c ", *b_current(b) );
            }
            else
                printf("something's wrong\n");

            if( b_ateol(b) )
            {
                if( b_ateob(b) )
                    printf("    <- Last line");
                printf("\n");
            }
        }
    }
    b_setrc(b, startrow, startcol);
}
/*-------------------------------------------------------------------*/
void valid_test( textbuf *b )
{
    int row, col;
    printf("\nThese indexes (r,c) are invalid in a %d by %d buffer:\n",
                                        b_nrows(b), b_ncols(b) );
    for( row = -1; row <= NROWS; ++ row )
    {
        for( col = -1; col <= NCOLS; ++ col )
            if( !b_valid( b, row, col ) )
                printf( "(%2d,%2d) ", row, col );
        printf("\n");
    }
    printf("\n");
}
/*-------------------------------------------------------------------*/
void crlf_test( textbuf *b )
{
    printf("Issuing LF:\n");
    b_lf( b );
    printbuf(b, 0);

    printf("\nIssuing CR:\n");
    b_cr( b );
    printbuf(b, 0);
}
/*-------------------------------------------------------------------*/
void clear_test( textbuf *b )
{
    printf("\nResetting cursor to (1,3):\n");
    b_setrc(b, 1, 3 );
    printbuf(b, 0);

    printf("\nDeleting three characters, using '.' fill:\n" );
    b_delete_c(b, 3, '.' );
    printbuf(b, 0);

    printf("\nInserting three characters, using '.' fill:\n");
    b_insert_c( b, 1, '.' );
    b_insert_c( b, 2, '.' );
    printbuf(b, 0);

    printf( "\nClearing entire line, '-' fill\n" );
    b_clearline( b, '-' );
```

continued



Avoid screen building headaches with.....SMP

- ☑ Windows
- ☑ Menus
- ☑ Mouse Support
- ☑ Keyboard Support
- ☑ Context Sensitive Help
- ☑ Low RAM Overhead / High Speed

Screen Manager Professional, the advanced interface design library for C programmers, gives your applications:

To order SMP call:

Magee Enterprises, Inc. 1-800-662-4330
Demo Available 404-446-0271
BBS 404-446-6650

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    printbuf(b, 0);

    printf( "\nDeleting line 1, moving top text down, '+' fill:\n");
    b_setrc(b, 1, 3 );
    b_closeup(b, '+');
    printbuf(b, 0);

    printf( "\nDeleting top line, moving bottom text up, '*' fill:\n");
    b_setrc(b, 0, 3 );
    b_closedown(b, '*');
    printbuf(b, 0);

    printf( "\nOpening top line, pushing text down, '%' fill:\n");
    b_opendown(b, '%');

    printbuf(b, 0);

    printf( "\nOpening middle line, pushing text up, '#' fill:\n");
    b_setrc(b, 1, 3 );
    b_openup(b, '#');
    printbuf(b, 0);

    printf( "\nClearing buffer, '@' fill:\n");
    b_clearbuf(b,'@');
    printbuf(b, 0);
}
/*-------------------------------------------------------------*/
main()
{
```

continued

A call to the macro **b_closedown(b,'*')** would result in:

```
 a  b  c  d  e  f  g  h
(q) r  s  t  u  v  w  x
 *  *  *  *  *  *  *  *
```

ADDING LINES

The complementary open-line macros are **b_openup(b,c)** and **b_opendown(b,c)**. The former opens up a new line on the current row by moving both the current line and all text above the current line up one notch. The previous top line is lost and the new line is filled with character *c*. The cursor ends up at the far left of the new line. Given the earlier initial buffer, a call to **b_openup(b,'*')** yields:

```
 i  j  k  l  m  n  o  p
(*) *  *  *  *  *  *  *
 q  r  s  t  u  v  w  x
```

The macro **b_opendown** does the same, but the text beneath the line is pushed down. Again, starting with the original buffer, a call to the **b_opendown(b,'*')** macro yields:

```
 a  b  c  d  e  f  g  h
(*) *  *  *  *  *  *  *
 i  j  k  l  m  n  o  p
```

Listing 7 contains a simple test program which exercises the various **textbuf** subroutines and macros. It also demonstrates how the different macros work and has some sample program output. Listing 8 shows you partial output from the test program.

UNTIL NEXT TIME

This first half of my two-part article on windowing in Turbo C has examined that section of the window system that manipulates underlying text. The second article will describe how to add a display layer to support the viewport and also how to manipulate the viewport on top of the text layer. PJ

Turn your favorite C compiler into a powerful database manager with the
C/Database Toolchest
The C/Database Toolchest™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the C/Database Toolchest™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.
The C/Database Toolchest™ includes three major components:
1) An advanced B+tree library gives you instant access to your data.
2) A high-level ISAM library provides you with an easy-to-use C interface, and
3) A complete database manager (with C source code included) shows you how to create impressive applications.
You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.
The C/Database Toolchest™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.
About the only thing that the C/Database Toolchest™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.
Now Only $19.95!
Order Coupon
Name
Street
City
State Zip
Telephone
Paying By Money Order Check
Visa MC AX Disc.
Card#
Exp. Date
Disk Size 5¼" 3½"
Qty Product Price Subtotal
C/Database Toolchest $19.95
C/Database Library Source $19.00
B+tree & ISAM library source code
Add Shipping ($5 USA, $20 Foreign)
Texas Residents Add 8% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001
MIX software
C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.
Circle 266 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

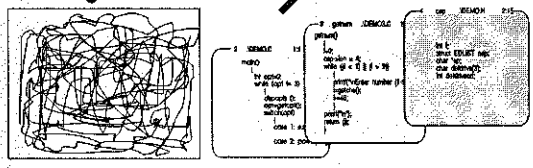
4c™ for Programmers

Analyze, Organize, Arrange

Your source code environment
With the 4c Hypertext facility!

The 4c Hypertext facility saves you time and money.

Change This → to This



Here's how!

✔ Analyze:
The Analyzer maps and cross-references functions and variables, to their respective path and file names. This cross reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name. A great documentation aid!

✔ Organize: with our *ZOOM* key!
Which instantly locates any function, global, variable, structure definition, #define, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

✔ Arrange: with our editor.
Featuring multiple files, multiple windows, cut-and-paste between windows, search-and-replace, auto indent, tab support, and a familiar user interface with pull down windows, or use your own editor!

Versions Now Available:
4c - Classic (the original with built in editor)
4c - Brief (add 4c capability to Brief)
4c - Windows (Windows 3.0 version of 4c)

60 Day Moneyback Guarantee!
No copy protection!
Free Technical Assistance!
For C & Pascal Programming Languages.
No changes to source code required.
Mastercard, Visa, Cod, PO, or check accepted

Only $119 US funds plus shipping.

Call or Write Today!

Tri-Technology Systems, Inc.
1000 Jorie Blvd., Suite 52
Oak Brook, IL 60521
1-708-366-7595

Editor Developers:
Add 4c functionality to your Windows 3.0 editor! Call, regarding the 4c DDE interface specification for your editor.

Circle 276 on Reader Service Card

Program FASTER with ...

CC-RIDER

or the new **C++ RIDER**

POP-UP SOURCE BROWSERS FOR ANY EDITOR !

CROSS-REF Editing!

Instantly pop-up your symbol definitions with a *hypertext hot-key* inside any editor! The *Standard Edition* has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all *uses* of a symbol in your code as you edit ! C++ RIDER fully supports C++ v2.1, including classes, constructors, overloading, etc.

Source Analysis!

Build a database of symbol information with our powerful ***source code analyzer*** which provides a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE AND HOW all your symbols are used. And it now uses expanded memory for ***lightning fast*** source code scanning!

QuickHelp, too!

CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.*** Works with ANSI/Microsoft C, plus AT&T/Zortech/ Borland C++.

For DOS and OS/2
$89 / 179 / 279

Western Wares

(303) 327-4898
Box C Norwood, CO 81423

FREE DEMO DISK

Circle 100 on Reader Service Card

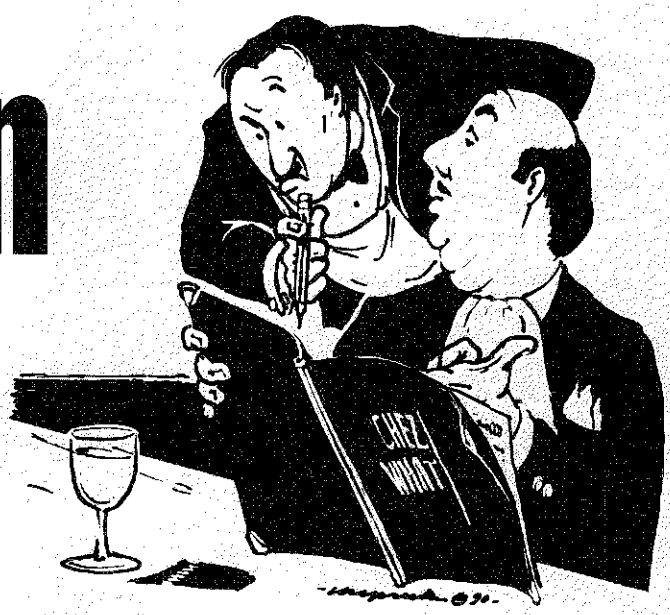

```
    int row, col, i;
    textbuf *b;

    if( !(b = b_new( NROWS, NCOLS )) )
        printf( "Can't allocate the textbuf\n" );

    /* Initialize buffer as if it's a 1-dimensional array.
     * It's initialized to:     a b c d e f g h
     *                          i j k l m n o p
     *                          q r s t u v w x
     */
    b_setrc(b,0,0);
    for( i = 0; i < NROWS * NCOLS; ++i )
        *b_advance( b ) = 'a' + i;

    b_setrc(b, 1, 4);
    printbuf( b, 1 );

    valid_test( b );
    crlf_test ( b );
    clear_test( b );

    return 0;
}
```

LISTING 8

```
/*texttext.out- Partial output from Test-Program */
 a  b  c  d  e  f  g  h
 i  j  k  l (m) n  o  p
 q  r  s  t  u  v  w  x     <- Last line

These indexes (r,c) are invalid in a 3 by 8 buffer:
(-1,-1) (-1, 0) (-1, 1) (-1, 2) (-1, 3) (-1, 4) (-1, 5) (-1, 6) (-1, 7)
(-1, 8) ( 0,-1) ( 0, 8) ( 1,-1) ( 1, 8) ( 2,-1) ( 2, 8) ( 3,-1) ( 3, 0)
( 3, 1) ( 3, 2) ( 3, 3) ( 3, 4) ( 3, 5) ( 3, 6) ( 3, 7) ( 3, 8)

Issuing LF:
 a  b  c  d  e  f  g  h
 i  j  k  l  m  n  o  p
 q  r  s  t (u) v  w  x     <- Last line

Issuing CR:
 a  b  c  d  e  f  g  h
 i  j  k  l  m  n  o  p
(q) r  s  t  u  v  w  x     <- Last line

Resetting cursor to (1,3):
 a  b  c  d  e  f  g  h
 i  j  k (l) m  n  o  p
 q  r  s  t  u  v  w  x     <- Last line

Deleting three characters, using ' ' fill:
 a  b  c  d  e  f  g  h
 i  j  k (o) p
 q  r  s  t  u  v  w  x     <- Last line

Inserting three characters, using '.' fill:
 a  b  c  d  e  f  g  h
 i  j  k (.) .  .  o  p
 q  r  s  t  u  v  w  x     <- Last line

Clearing entire line, '-' fill
 a  b  c  d  e  f  g  h
(-) -  -  -  -  -  -  -
 q  r  s  t  u  v  w  x     <- Last line

Deleting line 1, moving top text down, '+' fill:
 +  +  +  +  +  +  +  +
(a) b  c  d  e  f  g  h
 q  r  s  t  u  v  w  x     <- Last line

Deleting top line, moving bottom text up, '*' fill:
(a) b  c  d  e  f  g  h
 q  r  s  t  u  v  w  x
 *  *  *  *  *  *  *  *     <- Last line

Opening top line, pushing text down, '%' fill:
(%) %  %  %  %  %  %  %
 a  b  c  d  e  f  g  h
 q  r  s  t  u  v  w  x     <- Last line


/* The full code is available on the PJ 9.1 code disk. */
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

What's on the menu



A simple Lotus-style interface for your next database project

By M. Steven Baker

Nowadays probably the most difficult part of developing any new application is designing the user interface. Unless your spiffy new program is the equivalent of a powerful batch processor, you as the user will need to make choices and enter data. The interface design can make an application a joy to learn and use, or a chore. Back in the distant days of CP/M-80 and 64K, one could get by with the most horrendous user designs and still survive. Today, the user interface can literally make or break an application.

Unfortunately, many custom database programs still suffer from the most boring and unimaginative design of all—the classic pick-a-number menu tree. Although there is value in designing a custom interface from scratch, most of the world has neither the luxury nor the talent to do so. It's a good idea to build on the better designs already in existence. The most important secondary benefits of this approach are familiarity and a reduced learning curve for the user.

This article describes a simple and effective interface design I have used in several database applications. The concept is loosely based on the Lotus-style menu bar. Here I have implemented the parts of the 1-2-3 interface that seemed most valuable and left the rest to the litigators (since the PC corporate world seems bent on spending their excess profits suing each other in lieu of using them to work on developing superior products).

A SCREEN FOR ALL SEASONS

It all started when I was faced with constructing a database system on Oregon rivers (if you read *PJ* 8.6, you're already familiar with this project). I had little time for luxury, being faced with the task of developing an operational program *pronto*. In spite of the hurry, I insisted on first developing a prototype of the application. Back then, I used Dan Bricklin's DEMO program (now marketed as an improved DEMO II by Sage Software). This added a week's time, but in the development cycle it saved ten times that. Prototyping an interface continues to be the most valuable (and often neglected) step in developing a custom application. It doesn't matter what prototyping program you use (DEMO II, Proteus, Instant Replay, ProtoFinish, etc.): if you don't have one now, please purchase a decent prototyping tool and learn how to use it.

The custom database application was a data retrieval system on Oregon rivers. In this particular system, data file maintenance is handled at a central location while copies of the application and data files are used by any number of remote users on their own PCs. I decided to emulate some of the Lotus-style menu bar for several reasons: this interface is a well known metaphor for users; the basic menu system continually provides one line of help, mitigating the need (at least initially) for a full-blown help system; and the keyboard interface design allows maneuvering around with fewer keystrokes than any other interface.

Pull-down menu schemes like IBM's CUA or Borland's Turbo interface, on the other hand, generally require more keystrokes to accomplish the same actions. The Lotus keyboard interface design also supports both a quick key input (usually the first letter of a command) and the slower, more intuitive pick and selection mechanism (arrow keys that highlight the command with the ENTER key used for choice).

Since the application required a series of menu trees, I incorporated several other design features. Probably the most valuable one was displaying the current menu name at the top of the screen, so that users can always see where they are in the complex menu tree structures we tend to create. This feature resulted in the Lotus menu bar's being pushed to the bottom two lines of the screen, which worked out just fine.

For this particular application, a modest amount of geographic and location data was needed to identify a river segment, while an increasingly large amount

M. Steven Baker is the editor of Programmer's Journal *and a co-author of* Extending DOS, *published by Addison Wesley. On occasion, he can be found reluctantly manipulating large databases.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
*** MENUKEY.PRG
*   Author:     M. Steven Baker

private m_row,m_col
notfound = .T.
DO WHILE notfound
   IF (remenu)
      @ 23,0 CLEAR to 24,79     && clear menubar area
      @ 23,3 SAY menuline
      IF (menusel <1 .OR. menusel > menu_max)
         menusel = 1
      ENDIF
      SET COLOR TO N/W
      @ 23,VAL(SUBSTR(menubar(menusel),1,2)) ;
         SAY SUBSTR(menubar(menusel),3,LEN(menubar(menusel))-2)
      SET COLOR TO W/N
      @ 24,3 SAY menuhelp(menusel)
   ENDIF
   badkey = .T.
   DO WHILE (badkey)
      choice = 0
      DO WHILE (choice = 0)
         choice = inkey()
      ENDDO
      m_row = row()             && save cursor position
      m_col = col()
      @ 0,60 CLEAR to 0,78      && clear error line at top
      @ m_row,m_col SAY ""      && restore cursor position
      IF (choice >32 .AND. choice <127)  && an ASCII char?
         choice = UPPER(CHR(choice))
         m = 1
         DO WHILE (m <= menu_max .AND. badkey)
            IF ( SUBSTR(menuchoice,m,1) = choice)
               notfound = .F.
               badkey = .F.
               menusel = m
            ELSE
               m = m+1
            ENDIF
         ENDDO
      ELSE
         badkey = .F.
         DO CASE
            CASE choice = 4      && right arrow
               IF (menusel < menu_max )
                  menusel = menusel +1
               ELSE
                  menusel = 1
               ENDIF
               choice = SUBSTR(menuchoice,menusel,1)
            CASE choice = 19     && left arrow
               IF (menusel = 1)
                  menusel = menu_max
               ELSE
```

continued

of resource data would be requested for this river segment. For screen design I opted for a large boxed window with this identification data at the top. Below that, a smaller child window is swapped in with more detailed data on various resources. Field descriptions in these windows are shown in normal intensity, while actual displayed data is highlighted as "bright." Errors are reported in the upper right-hand corner above the boxed main window. Figure 1 shows a sample interface screen in which several of the parts have been noted.

INPUT, INPUT ON THE WALL

A single procedure (**MENUKEY**) is used for keyboard input (see Listing 1). This procedure uses several global variables to handle the menu bar, cursor keys, and menu option selection. **MENUKEY** ultimately uses the **INKEY** function to get single keystroke input from the underlying dBASE engine. The legal options for menu selection are passed in a string variable called, appropriately enough, *menuchoice*. The maximum number of menu options is passed in variable *menu_max*, and the actual menu bar line in *menuline*. Set these variables before calling **MENUKEY**, using something like this:

```
*------ our current MENU BAR line
menuchoice = "NPDUTRFQ"
menu_max = 8
menuline = "Next  Previous  Downstream  " + ;
   "Upstream  Trib  Resources  Flow  Quit"
```

MENUKEY needs information about the screen location of each possible selection option in this menu bar line, so the current option is highlighted in reverse video. Also, a single line of help is tied to each current menu option. Because dBASE III+ didn't support arrays at the time, this was handled by numbered variables. Two sets of parallel variables were used to emulate an array: *menubar1* through *menubar9*, and *menuhelp1* through *menuhelp9*.

In the example, the maximum possible number of menu bar selections is set at 10 (the array dimensions). This may be changed to whatever you desire. The *menu_max* variable is set to the maximum options for any particular menu line.

The *menubar* variables contain both the screen column offset and the string that should be highlighted in reverse video. Living within the dBASE III syntax, it would look like this:

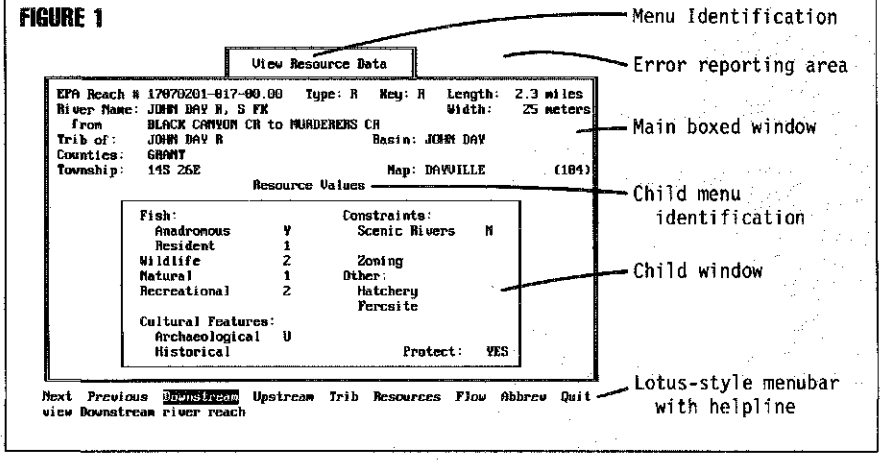
```
menubar1 = "03Next"
menubar2 = "09Previous"
* etc
menuhelp1 = "view Next river reach"
menuhelp2 = "view Previous river reach"
```

However, since almost every dBASE compatible engine (even dBASE IV) now supports arrays, a more compact coding scheme with array variables can be used. The code below gives an example using arrays.

```
menubar(1) = "03Next"
menubar(2) = "09Previous"
* etc
menuhelp(1) = "view Next river reach"
menuhelp(2) = "view Previous river reach"
```

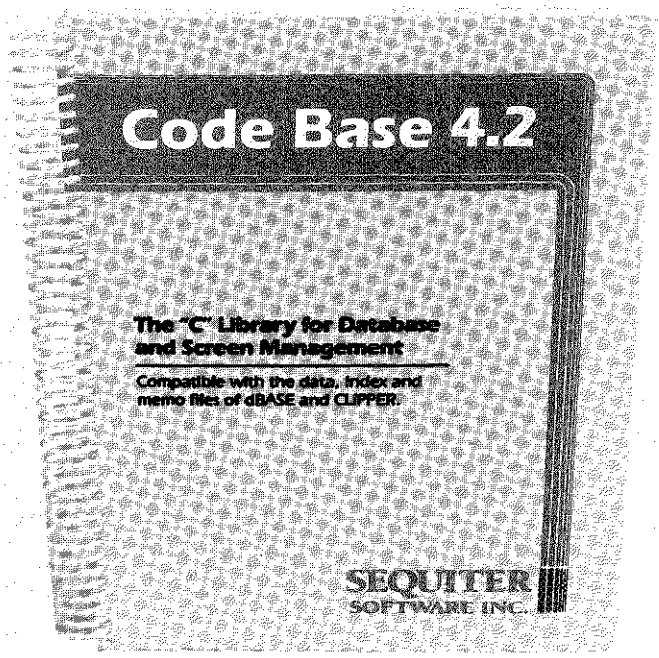
The first two characters of *menubar* correspond to the option string's starting

FIGURE 1



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Speed
C Portability
C Flexibility


Code Base 4.2
The "C" Library for Database and Screen Management
Compatible with the data, index and memo files of dBASE and CLIPPER.
SEQUITER SOFTWARE INC.

dBASE Power

Build a multi-user, dBASE compatible application which is several times faster than dBASE IV, Clipper or Fox Pro. Watch its windows and menus appear instantly on any computer.

Portable
Port your application to any environment with a C or C++ compiler. Access megabytes of memory using 386 DOS compilers, OS/2, Unix or Microsoft Windows.

Compatible
As you directly use the data, index and memo files of dBASE III through IV or Clipper, you can use Code Base 4.2 with any dBASE compatible product.

Easy
Consult examples in the 280 page user's guide as you interactively execute Code Base 4.2 routines from a learning utility. You will remember the routines which are named like dBASE commands.

Small
Make stand alone executable files as small as 14K. Code Base 4.2 executables are ½ to ⅓ the size of corresponding Clipper executables.

Complete
Enjoy the benefits of complete dBASE functionality, including browse, edit, menus, windows, multiple index files per database, dBASE expression evaluation, relations and filters.

Order Today
Order the DOS-OS/2 version for $295. Call (403) 448-0313 or fax (403) 448-0315. Discover why Sequiter Software Inc. and most software dealers offer a 60 day money back guarantee. Source is included and there are no royalties!

Sequiter Software Inc. • P.O. Box 5659, Station L, Edmonton, Alberta, Canada T6C 4G1

Circle 269 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

Opus Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- DOS version uses only 3K memory!
- Multiple directory support.
- Supports: Polytron PVCS™, Burton Systems TLIB™, Microsoft LIB™, SLR Systems OPTLIB™
- Generates automatic response files of unlimited length for LINK and LIB.
- OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or P.O. 60 day MBG!

Call 1-800-248-OPUS
International: 415-684-7091
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 254 on Reader Service Card

Experts Always Agree on the Solution!

However, they rarely agree on the features that make Salford's FTN77/386 superior!

"The defaults for the compiler are easily configured, and the compiler supports a load and go option for quick testing".
. . . Steve Baker - "Programmer's Journal", July 1990

"The debugger was very easy to use. There is an on-line help facility which made reference to the manual unnecessary".
. . . Mike Gunn and Arul Britto, ".EXE", May 1989

"FTN77/386 provides more intrinsic functions than any other: functions to manipulate string, handle screen and keyboard I/O, read the date and time, parse the command line . . . . Advanced file manipulation functions allows an FTN77/386 program to work with files in a C-like manner".
. . . Scott Ladd, "Computer Language", Nov. 1988

"For sheer speed and sleekness, the OTG compiler stands out like a Ferrari among Yugos".
. . . Jack W. Crenshaw, "Computer Language", May 1990

• Important Benchmark Results •

73% - of our new business in 1989 came through existing customer references
99.6% - satisfaction rating from our existing accounts
100% - guaranteed satisfaction

OTG SYSTEMS, INC.

Clifford, PA • (717) 222-9100 • Fax: (717) 222-9103

SERVICE, QUALITY AND INTEGRITY
FTN77/386 Developed by Salford Software Marketing

Circle 270 on Reader Service Card

```
                    menusel = menusel - 1
                ENDIF
                choice = SUBSTR(menuchoice,menusel,1)
            CASE choice = 13    && return
                choice = SUBSTR(menuchoice,menusel,1)
                notfound = .F.
            CASE choice = 32      && space bar - advance right
                IF (menusel < menu_max )
                    menusel = menusel +1
                ELSE
                    menusel = 1
                ENDIF
                choice = SUBSTR(menuchoice,menusel,1)
            CASE choice = 9       && tab key - advance right
                IF (menusel < menu_max )
                    menusel = menusel +1
                ELSE
                    menusel = 1
                ENDIF
                choice = SUBSTR(menuchoice,menusel,1)
            OTHERWISE
                badkey = .T.
                notfound = .F.
        ENDCASE
    ENDIF
ENDDO    && do while(badkey)
ENDDO   && while notfound
RETURN
```

LISTING 2

```
*** Sample code to illustrate using MENUKEY

PUBLIC menubar(10), menuhelp(10)
PUBLIC menuchoice, menusel, menuline, remenu, choice
****** The VIEW DATA MENU screen
repaint = .T.       && must repaint the whole screen
rewindow = .T.      && must repaint the window
remenu = .T.
menusel = 1

DO WHILE .T.
    IF (repaint)
        *------ clear windows
            SET COLOR to W/N
        clear
        @ 0,27 to 2,52 DOUBLE   && draw header box
        @ 2,3  to 22,76 DOUBLE
        @ 2,27 SAY CHR(202)
        @ 2,52 SAY CHR(202)

        *-------  write header in bold
        SET COLOR TO W+/N
        @ 1,31 SAY "View Resource Data"
        SET COLOR to W/N
    *------ write out some sample sub-titles at top
        @ 3,5   SAY "EPA Reach #"
        @ 4,5   SAY "River Name:"
        repaint = .F.
    ENDIF
    IF (remenu)
        SET COLOR TO W/N
    *---------- our MENU BAR line
        menuline = "Next  Previous  Downstream  Upstream  " +;
              "Trib  Resources  Flow  Quit"
        menuchoice = "NPDUTRFQ"
        menu_max = 8
        menubar(1)  = "03Next"
        menubar(2)  = "09Previous"
        menubar(3)  = "19Downstream"
        menubar(4)  = "31Upstream"
        menubar(5)  = "41Trib"
        menubar(6)  = "47Resources"
        menubar(7)  = "58Flow"
        menubar(8)  = "64Quit"
        menuhelp(1) = "view Next river reach (alphabetic by name)"
        menuhelp(2) = "view Previous river reach (alphabetic by name)"
        menuhelp(3) = "view Downstream river reach"
        menuhelp(4) = "view Upstream river reach"
        menuhelp(5) = "view the upstream Tributary river reach"
        menuhelp(6) = "view detailed Resource values for this river reach"
        menuhelp(7) = "view detailed Flow and water quality data"
        menuhelp(8) = "Quit to master menu"
    ENDIF   && remenu
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    badkey = .T.
    DO WHILE (badkey)
        do menukey
        badkey = .F.
        DO CASE
            CASE Choice = 'N'
            CASE Choice = 'P'
            CASE Choice = 'D'
            CASE Choice = 'U'
            CASE Choice = 'I'
            CASE Choice = 'R'
            CASE Choice = 'F'
*               DO something
                @ 0,60 SAY "Good Input-"+Choice
            CASE Choice = 'Q'
                RETURN TO MASTER
            OTHERWISE
                    badkey = .F.
        ENDCASE
    ENDDO     && do while(badkey)
ENDDO   && the main true loop
```

LISTING 3

```
*** a revised version of menukey that supports a hot-key to save
*** the current EPA RRN and name to a database called ortemp.dbf
*** with an alias of TMP

private m_seekok
m_seekok = .F.
** Add the following option to MENUKEY's main case statement
** to support a hot-key option
                CASE choice = 22    && insert or CTRL-V
                    m_row = ROW()
m_col = COL()
m_dbf = dbf()               && save current database
m_select = select()         && save current work area
m_select = chr(m_select+64) && convert to ascii
save screen to m_screen
m_temprrn = EPA->rrn
m_tempname = EPA->name
IF .not. m_editflg
    ?? CHR(7)
ENDIF
select TMP
IF m_editflg .and. m_editidx  && is file indexed by RRN
    seek m_temprrn
    if EOF()
        m_seekok = .F.
        skip -1
    else
        m_seekok = .T.
    endif
ENDIF
IF .not. m_seekok .or. .not. m_editidx
    append blank
    replace TMP->rrn with m_temprrn
    replace TMP->name with m_tempname
    replace TMP->rev_date with date()
ENDIF
IF (m_editflg)
    edit
ENDIF
SELECT &m_select.
restore screen from m_screen
@ m_row, m_col SAY ""    && restore cursor
badkey = .T.
notfound = .T.
```


World's Fastest Windows Linker

SLR Systems is **YOUR** choice for the fastest, most powerful linking technology available. Wherever you are headed, chances are we are there, or already working on it.

OPTLINK technology is designed to have no internal limitations. There is **NO** inner limit on the number of modules, libraries, publics, segments, etc.

OPTLINK/Windows

- Faster than any other linker; up to 20 times faster than LINK.
- Links larger applications.
- Debugs larger applications.
- Compatible with MS segmented linkers for generating Windows and 16-bit OS/2 .EXE and .DLL files.
- Supports Codeview 3.0 and Windows 3.0.
- 30-day Moneyback Guarantee.
- Only $250.

"The first time or two that you use OPTLINK/ WINDOWS you tend to waste all the time that it has saved you checking that it really did work. And it did work."
Rick Parson
Manager, Software Engineering
Corel Systems

"It's great! It's far better than MS-LINK. I'm definitely going to start recommending it to clients."
Michael Geary
Windows Consultant

"Anyone developing software for Microsoft Windows without using OPTLINK/Windows has too much time on his hands."
Phil Huber
Director of Software Development
Wonderware Software Development Corporation

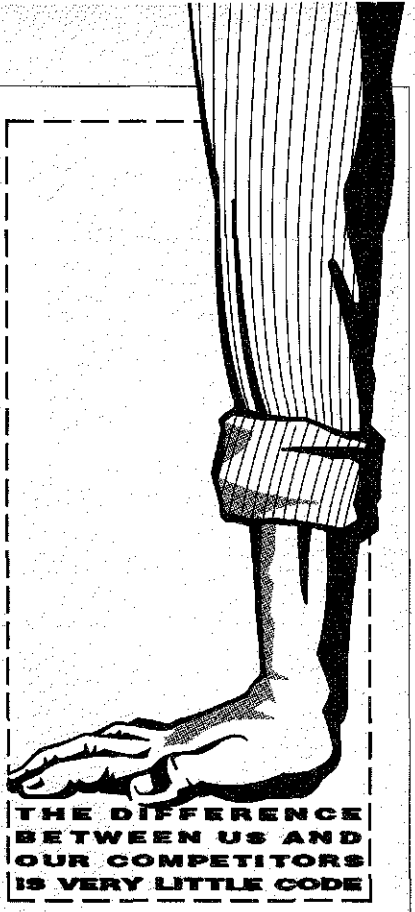
"OPTLINK/Windows took only 34 seconds to complete what took LINK 13 minutes. I'm very impressed. Your linker is over 20 times faster."
Peter Sharpe
Director of SGML Development
SoftQuad, Inc.

SLR SYSTEMS INC.

1622 N. Main Street, Butler, PA 16001 (412) 282-0864 Voice (412) 282-7965 FAX (412) 282-2799 BBS

Circle 165 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Magic TSR Toolkit

Write complete memory resident programs that require as little as 2K of RAM by swapping to EMS or disk. No secrets: complete assembly language source code and tutorial provided

Quantasm Power Lib

- Over 400 fast, compact assembly language routines
- Overlapping windows in less than 3K
- Over 100 string and numeric functions
- Menus, date, time, sound and more!

Magic TSR Toolkit . . .$199.95
Power Lib. 99.95
Power Lib w/source . . 299.95
ASMFLOW. 99.95

Shipping & Handling: U.S. $5; Foreign $15
CA res. add sales tax
VISA and MasterCard accepted
30-day money back guarantee

QUANTASM CORPORATION

19855 Stevens Creek Blvd., Suite 154
Cupertino, CA 95014
408/244-6826

Circle 252 on Reader Service Card

This isn't a "bug," it's another undocumented "feature."

column location. I based this scheme on an article I saw in *Databased Advisor* a few years ago. The article discussed user interface design with several dBASE tool packages, including QuickCode (Fox & Geller) and UI programmer (Wallsoft). The article described the interfaces and had a small sample of code to generate the interface. I never had access to QuickCode, but the article used a similar design for menu bar variables. Knowing that there are only so many ways to skin a cat, I proceeded with my design.

THE MARVELOUS MENUKEY

MENUKEY uses the position information to maintain the reverse video highlight over the correct selection. First the *menuline* is written to the screen, then the appropriate *menubar* entry is highlighted. This is handled by a *menusel* variable containing the current menu bar selection.

All of the logic that is required to maintain the menu and help lines is contained in this short **MENUKEY** procedure. Arrow keys, tab, space bar, and even ASCII input are handled completely by **MENUKEY**. When a legal selection is made, this key is returned in the global variable *choice*. (See Listing 2 for sample code.)

As an illustration of the power of this simple design, I'll include some code that extends **MENUKEY** to handle a special hot-key input. In my rivers database access system, a control-V or "insert" key would trigger this hot-key procedure. Listing 3 does the following:

- opens a secondary database
- appends a blank record
- adds several fields from the primary database
- opens a full-screen edit window

The hot-key feature uses a SETUP option from the main menu to configure the parameters. The current RIVERS setup option lets the user choose a secondary (temporary) dBASE file to use with the hot-key. In addition, it presents the option of indexing the second database so it will seek an existing record before appending one, a choice of the variable to index on, and other options. The hot-key feature also allows field users to add notes and to mark errors within this secondary database. I find the procedure useful for tagging and adding information to the database system. In short, what once was a simple data-retrieval system can now be used to locate an entry quickly and add new information as needed.

ARROW KEYS THAT WORK?

I originally developed this application four years ago in parallel versions for dBASE III+ and FoxBASE. If you're the sort who still insists on using outdated software, I must warn you that **MENUKEY** will not work properly under dBASE III+ without modification. The left-arrow key (control-S) does *not* work as it should unless combined with a series of software workarounds. Instead of returning a "19" as the Ashton-Tate manuals state, dBASEIII considers the left-arrow and control-S keys to be the xoff key in the xon/xoff communications protocol.

What a creative anachronism to lock up your screen! Under dBASE III+, escape processing must be turned off and restored to get the left-arrow key to function. I know—this isn't a "bug," it's another undocumented "feature." Fortunately FoxBASE, Clipper, and most of the other competitors lack this "feature."

Although the code I developed was limited to the dBASE syntax, I dropped testing under dBASE III+ years ago and switched to FoxBASE. There were two compelling reasons behind my decision to switch: fewer bugs and at least a threefold increase in speed. It's likely that Ashton-Tate would not be in its current state if the left-arrow key, **IIF** functions, etc. had worked properly, or at least had been fixed on a timely basis.

AND THE MEEK SHALL INHERIT

This procedure illustrates one way in which a powerful interface can be encapsulated in a very small amount of code. Based on thoughtful design, the example Lotus-style menu bar requires a surprisingly small overhead in both the **MENUKEY** procedure and in mainline code for interfacing. If only the new GUIs were so Spartan. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 DOS meets ANSI C

An overview of three 386 C compilers' conformance

By Rex Jaeschke

I started programming back in 1975, on a 16-bit PDP-11. While I got a lot of useful work done (and still do) on that machine, when it came to large projects I spent more time shoehorning my programs into the 64KB address space using the overlay linker than I did in writing the code to begin with. When the VAX was shipped in 1977, there was rejoicing in the streets because we now had a 32-bit address space and no one would ever need programs bigger than four gigabytes—right?

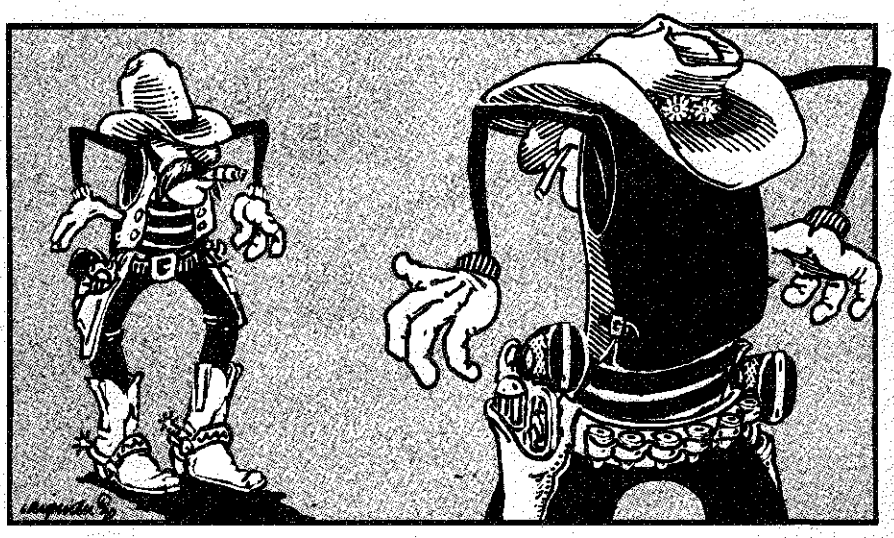
KLUDGES AND KLUDGES

Imagine my surprise, then, when IBM announced their first PC in late 1981. Yes, it had a 16-bit architecture—but it also had 20-bit addressing so you could accommodate programs of up to one megabyte. I groaned, along with many others. Once I discovered just what was involved in understanding memory models and, heaven forbid, how to mix them in the same program, I did the only sensible thing—I refused to believe this was more than a kludgy steppingstone on the path to a real machine. Accordingly, I avoided using anything but pure models; no near and far for me, thank you very much. Of course, the machine I was waiting for came in the guise of the 80386. However, there was more bad news: DOS didn't know how to use this new memory capability, so another (innovative and workable) kludge was invented: the DOS extender.

While I'm still waiting for real native support from my operating system, at least I can get some serious work done without being sidetracked by all this memory model nonsense. Well, almost. You see, the 386 does allow a segment to be as large as four gigabytes, so you don't need the usual near and far keywords, etc. However, the chip supports a 48-bit address space (for approximately four gazillion billion bytes, now isn't that enough for you!) yet has a 32-bit architecture. This means that if you want programs larger than four gigabytes, you'll need far 6-byte pointers. If you think this is way off the deep end and lightyears away, just read Watcom's and MetaWare's manuals: they already anticipate and support this possibility. The good news, though, is that for most of us mere mortals, a 32-bit address space will do just fine.

REAL MACHINES AT LAST

I have three DOS-class systems, including an original IBM-PC with a 64KB motherboard, and I know DOS well. In



Rex Jaeschke is an independent computer consultant, author, lecturer, and Editor of The Journal of C Language Translation. *Rex can be reached at: 2051 Swans Neck Way, Reston, VA 22091; (703) 860-0091; uucp: uunet!aussie!rex or aussie!rex@uunet.uu.net.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GRAPHICS LIBRARIES C/PASCAL/MODULA-2

Use the GFX libraries for all your graphics programming. They are small, full-featured, very fast, and have source code in your language.

Supports IBM PCs. Video cards - CGA, EGA, VGA (up to 800x600x16 colors) and Hercules.

GFX Graphics Library
Lines, circles, pie slices, boxes, polyline, filled polygons, XOR, rubber-banding. Integer and floating-point viewports with automatic scaling and clipping. Save and restore images (PCX).

$150 - Source Code

GFX Fonts & Menus Library
Over 100 bitmap fonts. Rotate and XOR characters. A slick graphics interface: mouse support; pull-down menus; dialog, selection and forms boxes.

$150 - Source Code

GFX Font & Icon Editor
This utility creates new characters and icons; removes and adds new characters or icons.

$100 - Source Code (C only)
$70 - Executable Utility

GFX Printer Screen Dump Lib
Dump graphics screen to your printer. Portrait or landscape; can preserve aspect ratio. Supports many printers: Epson, IBM, HP Laserjet & Paintjet.

$100 - Source Code

Compilers Supported
C - Microsoft, Borland, Lattice, Metaware, Zortech
Pascal - Borland
Modula-2 - JPI, Stoney Brook

Download demos/info from BBS
(816) 478-0944 : 2400/1200 Baud
8 Data/1 Stop/No Parity

Call us and mention this add for special pricing.

C Source
400 N.E. Point Dr.
Lee's Summit, MO 64064
Voice: (816) 478-1888
Fax: (816) 478-3133
Visa/MC accepted

Circle 336 on Reader Service Card

many of my introductory and advanced C seminars, I'm asked, "Just how can I squeeze this large program into 640K?" Until quite recently, my reply was "You don't! You write programs on a REAL machine." I'm happy to report that the 386 is a REAL machine and that DOS extenders really do let you exploit the architecture. The resources saved by not shoehorning large programs can now be applied to more useful endeavors such as debugging and adding extensions. Who knows, you might even have time to design and write documentation.

COMPILERS TESTED

The three 386-specific compilers I evaluated were:

- New C Version 1 for 386 (LPI)
- C Version 8 for 386 (Watcom)
- High C Version 2.3 for 386/486 (MetaWare)

All three of the tested compilers set the predefined macro **__STDC__** to 1 when ANSI mode is selected, indicating their claim to ANSI C conformance. My review is based on that claim.

The evaluation was conducted on a DELL 310 system, a 20MHz 80386 with 4MB RAM. MS-DOS V3.3 and Phar Lap's 386 DOS Extender V2.2d were used.

The compilers were configured either by using ANSI-mode options or, if none were explicitly available, by selecting non-extended mode. The maximum warning/checking level was enabled wherever possible.

Both Watcom C and High C support a compiler option to extract prototypes from existing function definitions.

NEW C FROM LPI

I found two preprocessor problems in New C: the identifier "defined" can be be the subject of **#undef** and **#define**; and an **#else** directive in a nested header is not diagnosed, as in:

```
#if ...
#include "header.h" /* contains lone #else */
#endif
```

A common problem with compilers is that a standard header often includes another standard header. This is very difficult (and usually impossible) to get right. For example, the identifier NULL is only reserved if you include one of the standard headers in which it is defined. If you don't include any of them, you are quite at liberty to use NULL for your own purposes. (Certainly this would be bad style, but it IS permitted by ANSI C.)

Therefore a standard header which includes *stdio.h* itself brings in the name NULL (and numerous other macros and typedef names as well) when this name is not defined as part of the first header. Yes, it can get messy to have each standard header be self-contained, but it can and must be done in strict ANSI mode.

The following are problems with the New C run-time library: *locale.h* includes *stddef.h*; *math.h* has various nonstandard names (HUGE, EDOM, ERANGE, and functions) and includes *errno.h* and *float.h*; and *signal.h* has the spelling of several dummy identifiers in prototypes in the users' namespace.

In addition, *stdio.h* includes both *stddef.h* and *stdarg.h*, and defines the nonstandard macro **OPEN_MAX**. It is surprising that **FOPEN_MAX** (the maximum number of files open at once) is only eight. Also, **FILENAME_MAX** (the maximum characters in a filename) is only 12. While eight concurrently open files may seem like a lot, DOS implementations typically support 20. And 12 is way too short for typical path names. The maximum size should be that supported by DOS.

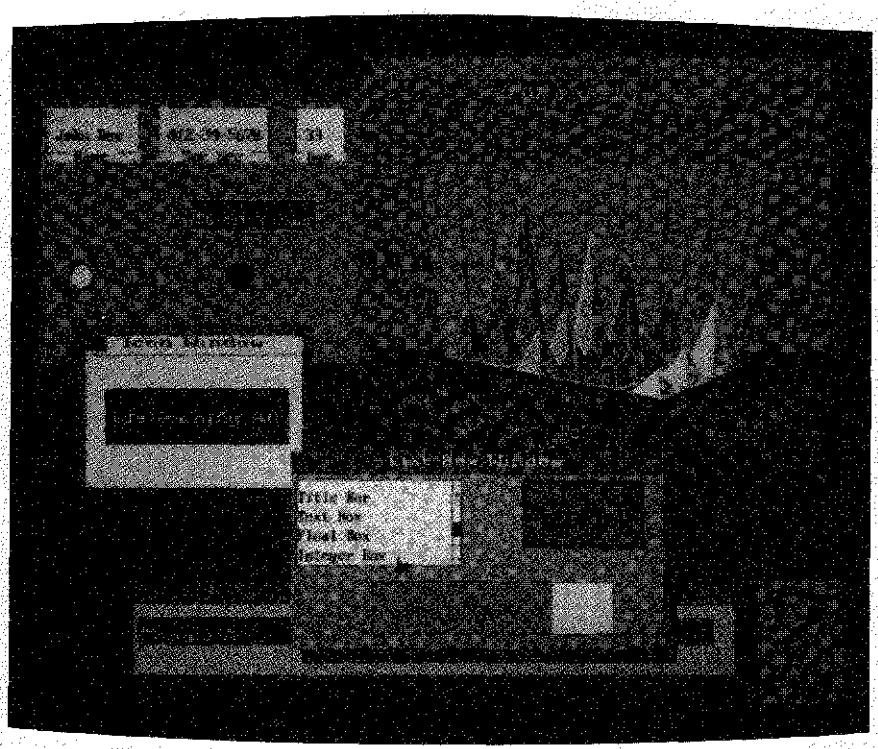
The *stdlib.h* file includes *stddef.h* and *limits.h*, and declares the nonstandard function **on_exit**; *string.h* includes *stddef.h*; and *time.h* includes *stddef.h* and *limits.h*. It also defines the nonstandard macro **CLK_TCK**.

MORE WITH NEW C

Phar Lap's 386 DOS Extender is included in the New C kit. The documentation consists of one slipcase containing a ring binder and a spiral-bound quick reference guide. Only the bare essentials are included—there is no discussion of the 386 architecture or how to exploit it. Note that LPI's CodeWatch debugger is mentioned in various places but is NOT yet available for the DOS product. This is stated in the release notes. On the other hand, New C's implementation-defined behavior is quite well documented.

In order to execute or even to compile programs containing floating-point operations, you MUST have a floating-point coprocessor (either an 80287 or an

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



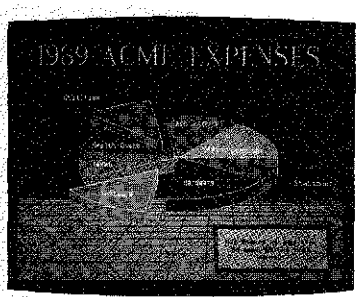
GRAPHICS TOOLS THAT SCALE NEW HEIGHTS

The true 3-D Charts, Data Entry Windows, List Boxes, and Help Screens pictured above are just the beginning. Below the surface you will find the two best collections of graphical programming tools available.

ESSENTIAL GRAPHICS LIBRARY™

The Complete C Graphics Function Library

The Essential Graphics Library is an extremely fast, powerful and reliable tool kit for creating today's demanding graphics applications. This complete graphics function library contains a full line of drawing and graphing tools, including true 3-D charting functions with world coordinates and viewports.



Add desktop publishing capabilities to your applications using built-in functions for font rotation and manipulation, along with PCX/PCC output. The library includes 12 bit mapped and 12 vector vector fonts and supports other font packages.

Compare these features:

- 3-D Bar, Pie, & Pyramid Charts
- 2-D Line, Scatter, High-Low-Close, & Bar Charts
- Super VGA, 8514, VGA, EGA, Hercules, & CGA Support
- Full Mouse Support
- Font Rotation & Scaling
- PC Paintbrush, PCX/PCC Image Support
- Dot Matrix, HP Laser and Plotter Output

If you are looking for a fast, reliable, well documented and fully supported library of C graphics tools, you just found it.

Graphical User Interface Dialog Objects

GUIDO™

Advanced Tools for Object Oriented Graphics Programming

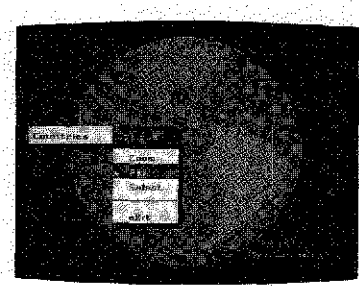
GUIDO is the object oriented C programming tool kit for creating today's graphics based user interfaces. GUIDO's Graphical User Interface Functions, with over a dozen predefined high level object types, supplies all menus, windows, boxes, buttons, and cursors. The Graphical User Environment provides an event driven, object oriented domain with full mouse and keyboard support, graphical screen handling and window overlap priority. Add multiple hot regions to each window and overlap an unlimited number of windows and objects.

No additional graphics library is required. All low level functions are included. GUIDO comes with a comprehensive manual with demo code and examples.

Compare these features:

- Pop-Up and Pull-Down Menus
- Vertical and Horizontal Menus
- Data Entry Objects
- List Boxes and Scroll Bars
- Radio Buttons
- Boolean Objects
- Pop-Up Text Windows and Percent Bars
- Custom GUI Objects
- Full Mouse and Keyboard Access

If graphic based object oriented programming is where you are headed, GUIDO will guide you to the top of the applications peak.



Order your copies today: EGL: $399 ($1399 with Source); GUIDO: $249 ($499 with Source). Try them for 30 days. If you are not completely satisfied return them in the original packaging for a full refund.

To Order Call 1-800-451-6174

*Six more essential programming tools from South Mountain: C Utility Library™, Essential Communications™, BreakOut-II™, /*resident_C*/™, and Hold Everything™. All libraries include support for Microsoft C/Quick C and Borland Turbo C.*

SOUTH MOUNTAIN SOFTWARE has supplied the highest quality, most up to date C programming tools and libraries for over five years. Every product comes with a 30 day, money back, guarantee. We provide free telephone support from experienced programmers, and you never pay a run-time fee or royalty to South Mountain. Our goal is simple, to give you the best C programming tools on the market today.

SOUTH MOUNTAIN SOFTWARE

76 South Orange Ave., South Orange, NJ 07079
Tel: 201) 762-6965 Fax: 201) 762-0118

Circle 10 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

80387 will do). Attempting to compile or execute such a program without a floating point coprocessor causes the system to lock up, requiring a complete power down and reboot. Since I do not have a coprocessor and my test suite has numerous floating-point tests scattered throughout, this was a constant source of aggravation. LPI has promised to look into providing a floating-point emulation library.

New C's listing file and cross-reference information are very good. Interlanguage calling conventions for other LPI language products are supported. The format and content of compiler messages is good. However, no pragmas are defined, and any pragmas that are seen are ignored.

New C is the first DOS C compiler I've seen that doesn't recognize quotes in command-line arguments as something special. For example, if you run a program using the command **RUN386 DOIT "A b"**, the command-line arguments received by **main** for the quoted string are the two arguments "*A* and *b*", rather than the one argument *A b* with the quotes removed. This behavior is acceptable as far as ANSI conformance goes because command-line issues are outside the scope of ANSI C. However, I did find it a bit unusual.

WATCOM C

As in New C, the Watcom C preprocessor fails to diagnose an **#else** directive in a nested header. Two C language problems were discovered: structures and unions can be assigned when they contain **const** members, and a tentative definition at file-scope level of a static object, such as **static int k[ ]**, is not diagnosed.

In the run-time library, *stdio.h* declares nonstandard functions **fcloseall**, **fdopen**, and **flushall**. *stdlib.h* declares nonstandard functions **fcvt**, **gcvt**, **swab**, **utoa**, and **ultoa**. It also includes the macros **MIN** and **MAX** and the identifiers **sys_errlist** and **sys_nerr**.

Moving to more general comments: the documentation consists of one very wide slipcase stand with seven spiral-bound manuals. There is a lengthy discussion of the 386 architecture and how to exploit it. Implementation-defined behavior is thoroughly documented.

The Watcom C kit includes a host of tools and libraries, including a debugger, a profiler, an editor/workbench, a **make** utility, and a graphics library. However, a 386 DOS extender is NOT included in the kit. Inline code and emulation libraries are both provided for floating-point operations.

THE POSITIVES

There are probably more pragmas included with Watcom C than you'll ever find use for. Among other things, they support custom-designed inter-language calling and optimizations. You can produce ROMable code and code that will support Windows, OS/2, QNX, and Novell's NetWare. (As a matter of fact, Novell's NetWare C compiler is Watcom's, repackaged.) The format and content of compiler messages is good.

Watcom provides their own linker rather than having you use 386LINK. A direct gain of this is that you can significantly reduce the size of executables on disk in certain cases. Consider the following program:

```
#include.h
main() {
    static char c[1000000];
    int i;
    for (i = 0; i  sizeof(c); ++i) {
        c[i] = 'x';
        if ((i % 100000) == 0)
            printf("i = %d\n", i);
    } }
```

The size of the programs resulting from this fragment is shown in Table 1. ANSI C requires that static objects be allocated memory and initialized prior to **main**'s being called. While many implementations do this at compile and/or link-time, Watcom does it at program startup, making the executable quite small. Note that such size reduction is possible because the static array has the default value of zero. This allows memory to be initialized in a loop at startup. If each element in the array had an explicit non-zero initial value, no such savings would be possible.

METAWARE HIGH C

The High C preprocessor has several problems. It permits **#undef** of the predefined macros __xxxx__ and of the identifier "defined." The # operator misbehaves when it is used with unusual macro calls. The preprocessor allows **#define defined(a)** .... In addition, the preprocessor does not complain about a non-terminated comment in a header. An **#else** directive in a nested header is not diagnosed (as in the other compilers tested). It also doesn't like **FILE** being a macro in the directive **#line LINE FILE**.

TABLE 1

Compiler	Executable file size
New C	42,972
Watcom C	8,436
High C	1,023,756

High C has a couple of C-language problems: it allows structures and unions to be assigned when they contain **const** members, and **long float** is accepted (with warning) as being equivalent to **double**, when it should be rejected.

Like the other compilers, High C has problems with its run-time library: *assert.h* includes *stdio.h* and *stdlib.h* to get at **fprintf**, **stderr**, and **abort**. In *float.h*, some **FLT*** macros have type **double** and should be type **float** instead. Several macros in *limits.h* include casts, but casts aren't permitted in **#if** directives, where these macros can be used. *stdio.h* includes *stdarg.h* to get a definition for the **va_list** type.

METAWARE MISCELLANY

A 386 DOS extender is NOT included in the High C kit. The documentation consists of four slipcases, each containing a ring binder and one or more manuals. There is a lengthy discussion of the 386 architecture and how to exploit it. Implementation-defined behavior is well documented.

The kit includes a host of tools and libraries such as debugger, profiler, editor, and graphics and utility libraries. Both inline code and emulation libraries are provided for floating-point operations. The type **long double** is mapped to the 80-bit IEEE extended precision when an Intel fp chip is used. The Weitek chip family is also supported, as is ROMable code development.

Many pragmas are included with High C. The listing file and cross-reference information is very good, as is the format and content of compiler messages. The compiler can load itself into extended/expanded memory so that larger source files can be accommodated. Unlike Watcom C and New C, High C creates relatively large executables.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I did have one problem during testing after installation. I traced it to the fact that the Windows V3 *smartdrv.sys* driver was installed, which did not leave sufficient memory for the compiler. After I removed the Windows driver, the compiler ran fine. Obviously, you would want both driver and compiler to coexist; I'm assuming the *smartdrv* installation parameters can be changed to allow this to happen. (My *smartdrv* was installed with default attributes as determined during its installation.)

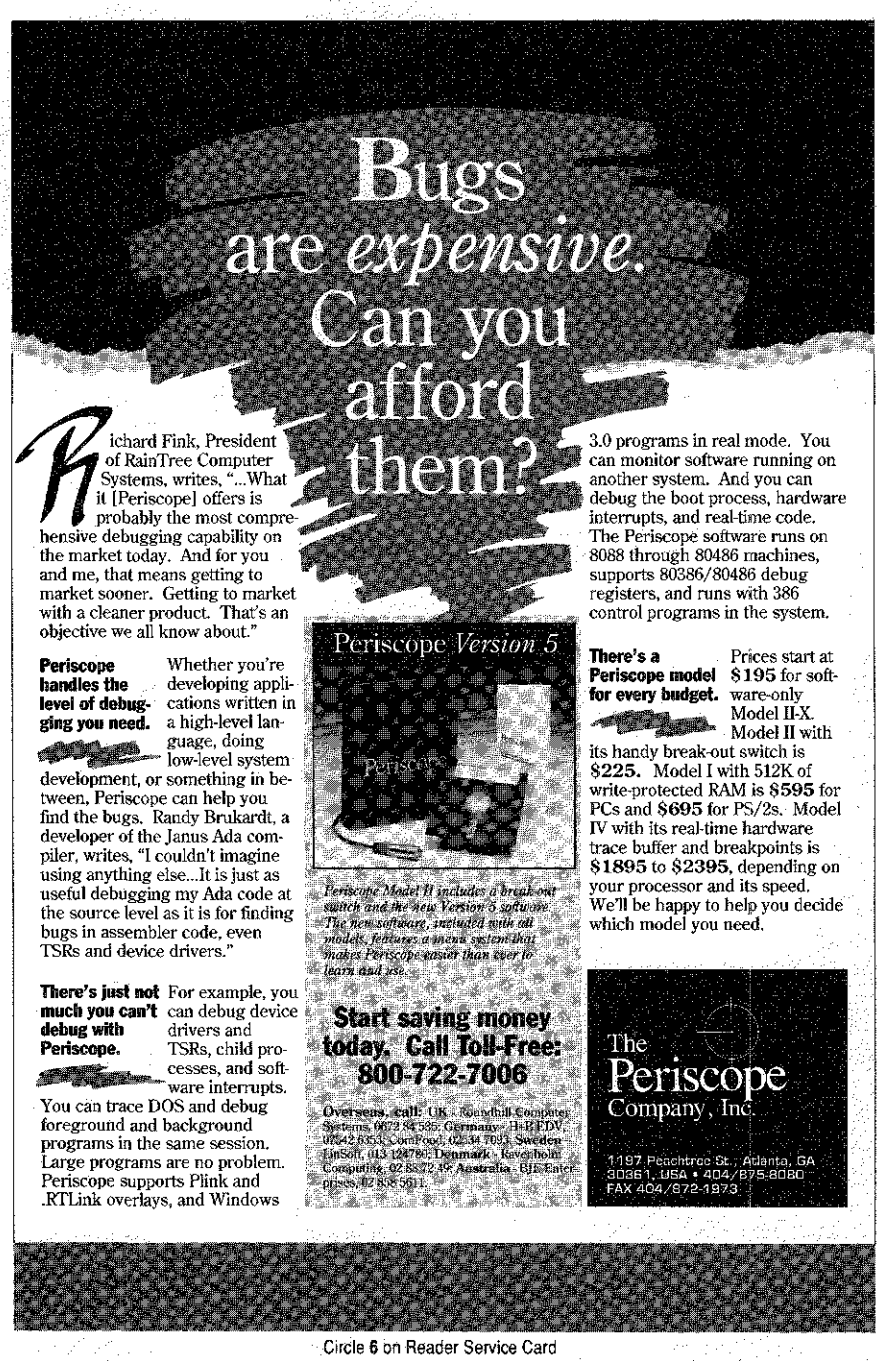
THE BOTTOM LINE

All three of the compilers I evaluated failed at least one of the tests in my conformance suite. However, the vendors' claims to ANSI conformance are justified in that they have made an honest attempt to make them compliant and, except for a handful of reasonably obscure problems, they do achieve conformance. In fact, the average programmer would likely be hard pressed ever to run into any of the shortcomings that I've mentioned above.

All of the three vendors are established producers of compilers for C and various other languages on numerous platforms. LPI is the new kid on the block as far as DOS is concerned, and it shows. But even though New C has less documentation and fewer support tools than the other two compilers do, it's still a very credible first offering. I've no doubt that it will improve. On the other hand, Watcom and MetaWare have further consolidated their already excellent reputations in the DOS/C marketplace, and will require some catching up to in the 386 arena. PJ

PRODUCTS TESTED

- New C Version 1 for 386, Language Processors, Inc., 959 Concord Street, Framingham, MA 01701-6933; (508) 626-0006.
- C Version 8 for 386, Watcom, 415 Phillip Street, Waterloo, Ontario Canada N2L 3X2; (519) 886-3700.
- High C Version 2.3 for 386/486, MetaWare Inc., 2161 Delaware Avenue, Santa Cruz, CA 95060-5706; (408) 429-6382.



Richard Fink, President of RainTree Computer Systems, writes, "...What it [Periscope] offers is probably the most comprehensive debugging capability on the market today. And for you and me, that means getting to market sooner. Getting to market with a cleaner product. That's an objective we all know about."

Periscope handles the level of debugging you need. Whether you're developing applications written in a high-level language, doing low-level system development, or something in between, Periscope can help you find the bugs. Randy Brukardt, a developer of the Janus Ada compiler, writes, "I couldn't imagine using anything else...It is just as useful debugging my Ada code at the source level as it is for finding bugs in assembler code, even TSRs and device drivers."

There's just not much you can't debug with Periscope. For example, you can debug device drivers and TSRs, child processes, and software interrupts. You can trace DOS and debug foreground and background programs in the same session. Large programs are no problem. Periscope supports Plink and .RTLink overlays, and Windows 3.0 programs in real mode. You can monitor software running on another system. And you can debug the boot process, hardware interrupts, and real-time code. The Periscope software runs on 8088 through 80486 machines, supports 80386/80486 debug registers, and runs with 386 control programs in the system.

Periscope Model II includes a break-out switch and the new Version 5 software. The new software, included with all models, features a menu system that makes Periscope easier than ever to learn and use.

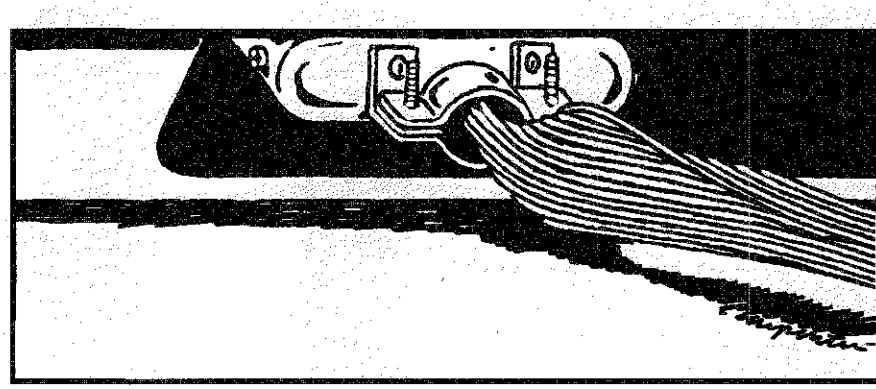
There's a Periscope model for every budget. Prices start at $195 for software-only Model II-X. Model II with its handy break-out switch is $225. Model I with 512K of write-protected RAM is $595 for PCs and $695 for PS/2s. Model IV with its real-time hardware trace buffer and breakpoints is $1895 to $2395, depending on your processor and its speed. We'll be happy to help you decide which model you need.

Start saving money today. Call Toll-Free: 800-722-7006

Overseas, call: UK - Roundhill Computer Systems, 0672 54 585; Germany - HiTEDV, 02642 6553; ComFood, 02 34 7093; Sweden - EmSoft, 013 124780; Denmark - Ravenholm Computing, 02 88 72 49; Australia - BJB Enterprises, 02 858 5011.

The Periscope Company, Inc.
1197 Peachtree St., Atlanta, GA 30361, USA • 404/875-8080
FAX 404/872-1973

Circle 6 on Reader Service Card



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Optimania

Using inlining and tail-recursion elimination

By Mark J. Barrenechea

PC-based compilers have matured in many respects, and particularly in the area of optimization. Three years ago the number of compiler vendors offering global optimization technologies was small; today, all popular PC compilers advertise that they are globally optimizing. In fact, differentiating among current compilers on the basis of optimization is difficult, since all are functionally similar.

In this optimania frenzy, it is important to understand each optimization and how all of them inter- and intra-relate. Used incorrectly, optimizations can unwittingly de-optimize code.

What follows is an explanation of the uses of and relationship between two popular optimizations—inlining and tail-recursion elimination. A change in coding philosophy has precipitated these optimizations: while FORTRAN programmers write large procedures, C, C++, and Pascal programmers tend to write small, well-defined procedures. Predictably, inlining and tail-recursion elimination have greater benefits in C, C++, and Pascal than in FORTRAN.

INLINING

Inlining is a general-purpose optimization that replaces a call to a function with the logic that is contained inside that function. This avoids the overhead of handling the prolog and epilog code inside the called function and of register saving and restoring, but with the potential cost of increased code size. Inlining of functions can also increase the potential for other optimizations, such as loop optimizations (if the call is in a loop), copy propagation (if a call no longer breaks propagation), constant folding (if the parameter is a constant), and instruction scheduling (an optimization for the 486, 4167, and i860). Replacing a call with its instructions makes it possible to schedule these instructions, thus removing delays.

Inlining is not macro expansion, and the semantics of the relevant language must be preserved. Hence, inlining supports call-by-value in C and Pascal, call-by-reference in FORTRAN, and both conventions in C++. Inlining can be applied to any language that maintains well-defined concepts for functions and procedures (C, C++, Pascal, FORTRAN, and even BASIC).

The main variant among languages is the way inlining should be invoked: either by command-line toggles or by type qualifiers like *inline* and *_Inline*. (The *inline* qualifier is not a legal ANSI C X3J11 extension, but *_Inline* is.) How to invoke inlining is a cultural issue and will not be discussed here.

Various issues are important when considering whether or not to inline a function:

- Size vs. frequency of use
- Recursivity
- Exported, imported, or static function definitions
- Taking the function address
- Nesting the function
- Using **alloca** with the function

SIZE VS. FREQUENCY OF USE

In general, small functions that are used many times are the best candidates for inlining. The cost of executing prolog and epilog code and the cost of register saving and restoring must be weighed against the time spent in the body of the function. Determining the cost of register saving and restoring is somewhat complex. There is the explicit cost of doing pushes and pops of registers, but there is also the potential cost of inefficient register coloring and spilling because of the call. When the call is removed and the code is "inlined," register coloring and coalescing are very different problems. The programmer usually has the responsibility of wisely choosing the functions to be inlined; a good profiler may be able to aid you in this decision process.

RECURSIVITY

The problem with recursive functions lies in deciding how many times to inline them. By their nature, they are called

Mark J. Barrenechea is director of language development at MicroWay, Inc. in Plymouth, Massachusetts.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

frequently. There are two choices: the first is not to inline them at all, and the second is to inline them as many times as you think the function will be recursively called—but with the risk of an exponential growth in code size. The programmer should control the number of times to inline a recursive routine. Of course, if the recursion can be replaced with iteration, then the routine can be inlined easily (more on this later).

FUNCTION DEFINITIONS

It is important to understand just what it means to have a function definition exported, imported, or hidden (static). When a function definition is non-static (extern), its definition is exported. In the following case, **my_strcpy**'s definition is *exported*:

```
...
my_strcpy(){
   ...
   }
```

When a function is not defined but a declaration is present, the function definition is then said to be *imported*. Consider the imported definition of **strcpy** below:

```
#include <string.h>
...
foo(){
    strcpy(s1,s2,n);
    }
```

Static definitions have file scope and are neither exported or imported, as in the following:

```
static foo() {
   ...
   }
```

In this case, **foo**'s definition is said to be *hidden* (static).

SOME INLINING CONSTRAINTS

Given these three classes of function definition, inlining constraints become clearer. Functions that have their definition exported may be inlined but must still be defined. Functions that have their definitions imported may be inlined, as explained below. Functions that are static may be inlined, and the definition may be discarded if all calls to that function are inlinable.

A general problem with inlining is that a function definition may not be available at compile time. In the following example, the function **memcpy** is declared as external in STRING.H and its definition is not seen until link time:

```
#include <string.h>
...
foo(){
   memcpy(s1,s2,size); /*definition is imported*/
   }
```

A library routine, or, more generally, any routine whose definition is imported, cannot traditionally be inlined; after all, no code is available with which to inline **memcpy**. Most compiler vendors, including Watcom, Green Hills, Zortech, Lahey, and MetaWare, do not allow the inlining of imported routines. (MicroWay, however, does allow it.)

The following technique shows how to allow exported routines to be inlined. Create a library archive file when inlining is activated. As compilation progresses, store function definitions (exported routines) in a library file (.inl). Inline imported routines by retrieving their definitions from this library. These .inl libraries can be provided for each standard library that is distributed by a compiler vendor (e.g., libc, libp, libcxx, libf, libm, etc.).

Functions whose addresses are taken may be inlined. Functions that are called through pointers to functions may be inlined only if the pointer is declared to be *const*. Otherwise, there is no guarantee that the compiler will be able to resolve which function is being pointed to.

NESTING AND ALLOCA

Of the languages discussed, only Pascal is allowed to contain nested procedures. (However, MetaWare supports nested functions in C and inlines them according to Pascal rules.) In general, nested procedures that are contained within inlined procedures are also inlined. If the nested routine has to be defined, it is cloned (duplicated) and multiple copies of the cloned procedure may exist.

The function **alloca** allocates memory on the program stack rather than using the free store (the heap). Because of this, any routine that contains a call to **alloca** must be treated specially in relation to prolog and epilog code. Again, there are two choices. The first is to inline any function containing a call to **alloca**, using the parent function's stack for memory allocation. The second is to realize that the programmer is playing coding tricks, and choose not to inline that function.

TAIL-RECURSION ELIMINATION

Optimizing by tail-recursion elimination is useful in limited situations. Recursion is given a bad name by many software engineers because it can use too much stack space and because recursive routines are too hard to maintain and understand. On the other hand, many algorithms in common use today are best understood and implemented using recursion. To remove recursion from these programs would actually make them more difficult to maintain. Nikalus Wirth went so far as to say that no professional programmer would use a language without recursion, and this author agrees. Make sure you think carefully before using tail-recursion elimination.

Certain direct recursive calls can be replaced by iteration. When the last statement in a basic block of code is a recursive call to the same procedure (i.e., direct recursion), the block of code is said to be tail recursive. Tail-recursion elimination can be useful because recursion overhead can be removed while maintaining the readability and coding philosophies of recursion: no extra function calls, no saving and restoring of registers, no multiple execution of prolog and epilog code, and no operating system overhead with multiple stack frames, pointers, and return addresses.

Both of the following examples are considered tail-recursive. (The second example is taken from Aho, Sethi, and Ullman's *Compilers, Principles, Techniques, and Tools*.)

```
void foo(void) {
   if (x)
      return;
   foo();
   } /* end of foo */
```

and

```
void reset(void){
   if (lookahead == '+') {
       match('+');
       term();
       putchar('+');
       reset();
       }
   else if (lookahead == '-') {
       match('-');
       term();
       putchar('-');
       reset();
       }
   } /* end of reset */
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Since the recursive call is at the end of basic blocks of code, the tail-recursive call can simply be replaced by a jump to the beginning of the procedure. Transformation of the **reset** function is shown below:

```
void reset(void) {
L:      if (lookahead == '+') {
            match('+');
            term();
            putchar('+');
            goto L;
            }
        else if (lookahead == '-') {
            match('-');
            term();
            putchar('-');
            goto L;
            }
        } /* end of reset */
```

Tail-recursion elimination becomes somewhat more challenging when formal parameters are passed to the recursive routine. Let's take a look at using Euclid's algorithm in computing the Greatest Common Divisor (GCD) of two integers.

The GCD of *a* and *b* is found as the GCD of *b* and *r*, where *r* is the remainder of *a* divided by *b*, and when *b* is non-zero. If *b* is zero, we can simply use **gcd(a,0) = a**.

```
int gcd( int a, int b ) {
   return( b ? gcd( b, a % b ) : a );
         /* this is tail recursive */
   }
```

Using a less terse form of C, this function can be transliterated into:

```
int gcd ( int a, int b ) {
   int r;
   if (!b)
      return a;
   r = a % b;
   gcd( b, r );
   /* this is tail recursive */
   }
```

Consider the transformation with tail-recursion elimination:

```
int gcd( int a, int b ) {
   int r;
L: if (!b)
      return a;
   r = a % b;
   a = b;
   b = r;
   goto L;
   }
```

The benefit of tail recursion is that formal parameters do not necessarily need to be stored on the program stack (as is the case with *a* and *b*). It is rather trivial for a compiler to replace the formal parameters with local assignments.

INLINING AND TAIL RECURSION

Many optimizations are synergistic when used together; their collective results produce better-optimized code than if they are used individually. Inlining used with tail-recursion elimination is a good example.

The best way to envision this synergy is by looking at the example shown in Listing 1. This example incorporates creating a sorted binary tree and then printing the tree in a depth-first ordering (DFO). In a sorted binary tree, the subtree on the left contains nodes that are "less than" or "equal" to the root, while those on the right are "greater than" the root. DFOs are useful in many applications; **make** does a depth-first ordering of targets to determine the processing order, and compilers have to do depth-first orderings of basic blocks to process code generation.

Real-Time Multitasking

Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider:

MultiDos Plus™

Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. Supports the loading and execution of tasks in LIM 4.0 expanded memory. For more information please call for a free brochure.

Call our BBS for free demo software.

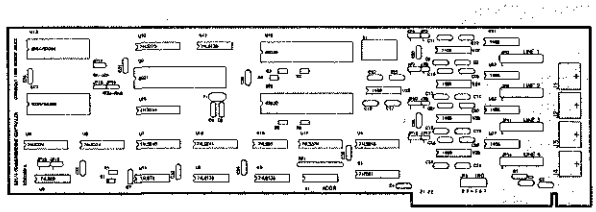
The complete software package is only $99 + S/H. Visa/MasterCard accepted.

30-day money back guarantee.
(subject to a $15 restocking fee)

Nanosoft Inc.
13 Westfield Road
Natick, MA 01760
800-678-2141
508-651-0091
FAX 508-655-8860
BBS 508-650-9552

Circle 109 on Reader Service Card

Intelligent Four Channel
Serial Communications Controller



- 8031 microcontroller
- 82530 SCCs
- 32K memory
- RS-232
- RJ-45 connectors
- DOS device driver
- FCC Class A approved
- Excellent documentation
- BASIC, C and Pascal software source examples
- **$495 complete** with all cables, software, & extensive documentation

Designed, Manufactured, Sold and Serviced by:

Rtech Inc.
Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . (612) 345-4656
907 N. 6th St., Lake City, MN 55041

Circle 36 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

In Listing 1, **stree** and **dfo** are typical procedures that are both tail-recursive. The procedure **initialize_tree** calls **stree** repeatedly with the next node for the tree retrieved by **get_next_node**. When no more nodes are available, it prints the tree in a depth-first ordering via **dfo**.

If 1000 nodes were entered into the tree, in the worst case 500,000 function calls would be required. With full inlining and with tail-recursion elimination turned on, zero function calls would be made. All recursion in **stree** is removed; in addition, the function can be inlined in **initialize_tree**.

The procedure **dfo** is particularly interesting because it contains two recursive calls to itself, the last of which can be replaced with iteration. If the tree is balanced, the function calls can be cut in half.

SOME INLINING PITFALLS

The most common misuse of inlining is non-selective inlining (i.e., inlining to the hilt). Given a large application, inlining all routines will more than likely be a mistake. Consider the following experiment. The MicroWay C++ compiler is approximately 300,000 lines of source code, 1600 procedures, and an executable size of 800K under Interactive UNIX. Recursing the compiler (having the compiler compile itself) takes 20.1 minutes.

When the compiler is recursed with full inlining, the executable size grows by about 16 percent and the resultant inlined compiler runs five percent slower. Thus, the compiler takes 21.2 minutes to compile itself.

Why is this? In many applications, high-level logic is controlled by omni-procedures—procedures that contain and maintain the flow of control, as in the following example:

```
main(){
   if some-condition
      do_pass_one(some-condition)
   else if another-condition
      do_pass_one(another-condition)
   else
      do_pass_one(no-condition)
   ...
   }
```

With full inlining on, **do_pass_one** will be inlined three times. The procedure **do_pass_one** has the potential of being very large, which will account for an explosion in code size. The degradation in speed can be attributed to poor register coloring. The procedure **main** is now much larger and register spilling may occur, whereas before, the graph was easy to color.

IT TAKES UNDERSTANDING

As compilers progress in global optimization technologies, their users must understand the inter- and intra-relationships among optimizations. Because compiler vendors allow users to control optimizations through command-line toggles and pragmas, mixing optimizations incorrectly can result in de-optimized code. In the software development phase of any application, understanding and controlling optimizations has become as important as writing structured code and employing good software engineering techniques. PJ

LISTING 1

```
Consider in C:

struct tree {
   char info;
   struct tree *left;
   struct tree *right;
   } node;

typedef struct tree TREE;

TREE *stree(TREE *root, TREE *cur, char info){
   if (!cur){
      if (!(cur = malloc(sizeof(node))) {
         printf("Out of memory\n");
         exit(0);
         }
      cur->left = cur->right = 0;
      if (info<root->info)
         root->left = cur;
      else
         root->right = cur;
      return cur;
      }
   if (info<cur->info)
      stree(cur,cur->left,info);  /* to left */
   else if (info>cur->info)
      stree(cur,cur->right,info); /* to right */
   }

void dfo( TREE *root ) {
   if (!root)
      return;
   dfo(root->left);
   printf("info -> %c ",root->info);
   dfo(root->right);
   }

void initialize_tree(void){
   while (!done)
      stree(root,get_next_node(),info)
   dfo(root);
   }
```


People are talking about us.

F77L-EM/32

Port 4GB mainframe programs to 80386s with this 32-bit DOS-hosted compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award is [illegible] better. New Version 3.0 and OS include: Editor, Make Utility, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $895 OS/386 $395

NEW RELEASE

F77L

The compiler of choice among reviewers and professionals. Includes a Debugger, Editor, Profiler, Linker, Make Utility, Weitek and 386 Real-Mode Support, Graphics. $595

Lahey Personal Fortran 77

New Version 3.0: Full ANSI 77, Debugger, Editor, Linker, Library Manager, Microsoft and Borland C interfaces, 400 page Manual, Unbeatable Price. $99

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

FORTRAN IS OUR FORTE

Circle 44 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Real-mode virtual memory

A file system with improved performance and more simplicity for assembly language programs

By John Otken

This column contains a high-performance file system which has been tailored for assembly language programs that read large files multiple times. The first goal of the system is performance, so it makes use of two techniques that affect performance—buffer sizing and memory management.

Buffer size is important because overhead on DOS file functions is significant. Large buffers reduce the number of DOS function calls and increase throughput. A 16K buffer is good for maximum speed and minimum waste. The size of an EMS page is also 16K.

BUFFER CACHING AND STORAGE

File system performance can be improved with the technique of buffer caching, as well. Rather than discard a file buffer on the next read, keep it around until memory is full. If you process the buffer again and it's still in memory, the program avoids one or two expensive DOS functions. Caching disk buffers can cause a problem, though: it quickly consumes memory until a program has no space for other storage. The solution is to merge the dynamic storage system and the file buffer cache.

John Otken is president of Soft Advances, where he hacks assembly language into debuggers and other software for embedded systems. Contact him at P.O. Box 184, Orinda, CA 94563.

A combined file and storage system manages 16K data objects known as *blocks*. A block can be a file buffer or dynamic storage. Blocks move between main memory, EMS, and disk. When memory becomes full, the block manager moves the least recently used (LRU) block to slower EMS resources. When EMS is full, the block manager moves the LRU block to even slower disk resources.

This system not only improves disk performance for some programs, it also implements a virtual memory system for dynamic storage blocks. A program can access many megabytes of storage and automatically execute faster on systems with greater resources (i.e., EMS).

KEEP IT SIMPLE

The second goal of this system is simplicity: the applications interface (API) must be easy to program. The file interface has two routines. Each 16K section of the file is "mapped" to a block with **block_file_open**. If the file is already "mapped," this routine returns the handle of the first block. The **block_file_read** routine returns a pointer to a block with the file's contents.

The storage interface also works with three routines: **block_alloc** returns a pointer and handle to a 16K storage block; **block_free** releases memory associated with a block; and **block_read** returns a pointer to a block. Notice that the interface routines use "handles" in

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; sample code - the full listing is on the PJ 9.1 code disk

        title   block storage manager
        include asm.inc

        public  block_alloc
        public  block_file_open
        public  block_file_read
        public  block_free
        public  block_read

BLOCK_MAX               equ     512
FIRST_BLOCK             equ     08000h  ; block handles start at 0x8000

NULL_BLOCK              equ     0
NULL_EMS                equ     0
NULL_ID                 equ     0
NULL_SWAP               equ     0

OPEN_LINK               equ     0FFFFh

block_str struc
  blk_forward           dw  ?   ; forward link for LRU
  blk_backward          dw  ?   ; backward link
  blk_pointer           dd  ?   ; RAM address (or 0 if in EMS or DISK)
  blk_ems_index         dw  ?   ; EMS index
  blk_swap_index        dw  ?   ; swap file index (0 if not swapped)
  blk_file_id           dw  ?   ; file identification (0 if not in file)
  blk_file_index        dw  ?   ; file block index
  blk_file_link         dw  ?   ; blocks for each file form a list
  blk_byte_count        dw  ?   ; (file blocks may be less than 16k)
block_str ends                  ; (structure must be even number of bytes)


XIB     segment word public 'DATA'      ; initialize block manager
XIB     ends
XI      segment word public 'DATA'
        dw      block_preset
XI      ends
XIE     segment word public 'DATA'
XIE     ends


        .const
ertx_block_handle       db      'Bad block handle',0
ertx_empty_file         db      'Cannot map empty file',0
ertx_free_block         db      'Cannot free block',0
ertx_past_eof           db      'Block read past EOF',0
ertx_virtual_full       db      'Virtual memory full',0


        .data?
block_table     dd      ?       ; array of block structures

ram_front       dw      ?       ; LRU list of RAM blocks
ram_back        dw      ?

ems_front       dw      ?       ; LRU list of EMS blocks
ems_back        dw      ?

free_front      dw      ?       ; free handle stack
free_count      dw      ?       ; number of free blocks

file_blocks     dw      ?       ; # of blocks in file (see block_file_open)

        .code
 extn abort_if_cf,calloc,clear_strerror,ems_exchange,ems_in,swap_in
 extn ems_out,free,malloc,save_most,set_strerror,swap_out,check_block_file
 extn open_block_file,read_block_file,set_block_handle

;;      access block table
;
;       entry   BX      block handle
;       exit    DS:SI   block table entry for handle
;               Cf      if bad handle
;       uses    AX
;
access_block_table proc
        mov     ax,bx
        sub     ax,FIRST_BLOCK
        jb      abt1                    ; if bad handle
        cmp     ax,BLOCK_MAX
        jae     abt1                    ; if bad handle
        mov     si,dx
        mov     dx,size block_str
        mul     dx
        mov     dx,si

        lds     si,block_table[bp]
        add     si,ax
        ret

abt1:   lea     ax,ertx_block_handle    ; *Bad block handle*
        jmp     set_strerror
access_block_table endp


;;      block alloc
;
;       exit    AX      block size in bytes
;               BX      block handle
;               ES:DI   block pointer
;               Cf      if no memory
;
block_alloc proc
        pushm   si,ds
        call    malloc_storage_block
        jc      bal2                    ;  if no memory

        call    new_block_handle
        jc      bal1                    ;  if no more block handles
        mov     wptr blk_pointer[si],di
        mov     wptr blk_pointer[si+2],es

        call    set_mru_ram             ; set most recently used
        mov     ax,BLOCK_SIZE
        mov     blk_byte_count[si],ax
        clc
        jmp     bal3

bal1:   call    free                    ; here when out of handles, free
bal2:   movx    bx,NULL_BLOCK           ;  storage and return error
        stc

bal3:   popm    ds,si
        ret
block_alloc endp


;;      block file open
;
;       entry   DS:SI   file name
;       exit    AX      file size in blocks
;               BX      block handle
;               Cf      if file not found or empty
;
block_file_open proc
        pushm   cx,dx,di,si,ds
        call    check_block_file
        jnc     bfo4                    ;  if existing file

bfo1:   call    open_block_file
        jc      bfo4                    ;  if file not found
        jcxz    bfo7                    ;  if empty file

        cmp     cx,free_count[bp]
        ja      bfo5                    ;  if not enough blocks for file

        mov     file_blocks[bp],cx
        movx    cx,0

        call    new_file_block          ; allocate first file block
        jc      bfo4
        mov     bx,ax
        cmp     cx,file_blocks[bp]
        je      bfo3                    ;  if file size <= 16k

bfo2:   mov     di,si
        call    new_file_block
        jc      bfo4
        mov     blk_file_link[di],si
        cmp     cx,file_blocks[bp]
        jb      bfo2

        call    set_block_handle        ; save handle in case we open again

bfo3:   mov     ax,file_blocks[bp]      ; return file size (Cf unchanged)
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
bfo4:   popm    ds,si,di,dx,cx
        ret

bfo5:   lea     ax,ertx_virtual_full    ; *Virtual memory full*
bfo6:   call    set_strerror
        jmp     bfo4
bfo7:   lea     ax,ertx_empty_file      ; *Cannot map empty file*
        jmp     bfo6
block_file_open endp


;;      block file read
;
;       entry   AX      16k file block number (0..n)
;               BX      block index
;       exit    DS:SI   block pointer
;               AX      block size (<16k if last block)
;               Cf      if bad block index or past end of file
;
block_file_read proc
        push    cx
        mov     cx,ax
        call    access_block_table
        jc      bfr4

        jcxz    bfr2                    ;  if first block
bfr1:   mov     si,blk_file_link[si]    ;  else chase linked list to
        cmp     si,OPEN_LINK            ;    selected blk
        loopne  bfr1
        je      bfr3                    ;  if block number too large

bfr2:   pop     cx
        jmp     block_read_primitive

bfr3:   lea     ax,ertx_past_eof        ; *Block read past EOF*
        call    set_strerror
bfr4:   pop     cx
        ret
block_file_read endp


;;      block free
;
;       entry   BX      block index
;       exit    Cf      if bad index (cannot free file blocks)
;       uses    AX,BX
;
block_free proc
        pushm   di,si,ds,es
        call    access_block_table
        jc      bfe5

        cmp     blk_file_id[si],NULL_ID ; illegal to free a file block
        jne     bfe6                    ;  if file block

        les     di,blk_pointer[si]
        mov     ax,es
        or      ax,di
        jz      bfe1                    ;  if not in RAM

        call    unlink_ram_list         ; unlink block from RAM LRU list
        call    free                    ; free storage
        jmp     bfe3

bfe1:   mov     ax,blk_ems_index[si]
        cmpx    ax,NULL_EMS
        je      bfe2                    ;  if not in EMS

        call    ems_in                  ; free EMS block (ES:DI==NULL)
        call    unlink_ems_list         ; unlink block from EMS LRU list
        jmp     bfe3

bfe2:   mov     ax,blk_swap_index[si]
        cmpx    ax,NULL_SWAP
        je      bfe6                    ;  if not in swap file (thus error)

        call    swap_in                 ; free swap block (ES:DI==NULL)

bfe3:   mov     bx,size block_str       ; clear block table entry and
bfe4:   dec     bx                      ;  leave BX==0
        dec     bx
        mov     wptr [si+bx],ZERO
        jnz     bfe4

        mov     ax,si                   ; link onto free block stack
        xchg    ax,free_front[bp]
        mov     blk_backward[si],ax
        inc     free_count[bp]

        clc
bfe5:   popm    es,ds,si,di             ; (POP xS may fault in protect mode)
        ret

bfe6:   lea     ax,ertx_free_block      ; *Cannot free block*
        call    set_strerror
        jmp     bfe5
block_free endp


;;      block preset
;
;       uses    AX
;
block_preset proc
        call    save_most               ; allocate block table
        mov     cx,BLOCK_MAX*size block_str
        call    calloc                  ;  (must have NULL offset)
        call    abort_if_cf
        mov     wptr block_table[bp],di
        mov     wptr block_table[bp+2],es

        mov     ax,OPEN_LINK            ; build free list
        mov     cx,BLOCK_MAX
        mov     free_count[bp],cx
        lds     si,block_table[bp]
        mov     free_front[bp],si
bps1:   lea     dx,[si+size block_str]
        mov     blk_backward[si],dx
        mov     blk_forward[si],ax
        mov     blk_file_link[si],ax
        xchg    dx,si
        loop    bps1
        mov     blk_backward[si-size block_str],ax

        mov     ram_front[bp],ax        ; initialize LRU lists for RAM/EMS
        mov     ram_back[bp],ax
        mov     ems_front[bp],ax
        mov     ems_back[bp],ax
        ret
block_preset endp


;;      block read
;
;       entry   BX      block index
;       exit    AX      block size
;               DS:SI   block pointer
;               Cf      if bad block or no memory
;
block_read proc
        call    access_block_table
        jnc     block_read_primitive    ;
        ret                             ;
block_read endp


;;      block read primitive
;
;       entry   DS:SI   block table pointer
;       exit    AX      block size
;               DS:SI   block pointer
;               Cf      if bad block or no memory
;
block_read_primitive proc
        pushm   di,es
        cmp     wptr blk_pointer[si+2],NULL_POINTER
        je      brp1                    ;  if block is not in RAM
        call    reset_mru_ram           ; move to front of LRU list
        jmp     brp5

brp1:   call    malloc_storage_block    ; allocate storage for block
        jc      brp6                    ;  if no memory

        mov     ax,blk_ems_index[si]    ; check if block swapped to EMS
        cmpx    ax,NULL_EMS
        je      brp2                    ;  if not in EMS

        call    ems_in                  ;  else swap block in from EMS
        jc      brp8                    ;   if unexpected EMS error
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

order to identify blocks. This is necessary because blocks can appear anywhere in memory. A program must never save the address of a block: instead, it should save the handle. When you need a pointer, call **block_read** with the handle.

BLOCK MANAGEMENT

While memory is plentiful, the block manager allocates 16K file and storage blocks. When memory runs out, the block manager moves the least recently used block to EMS, frees its memory, and allocates the memory for the new block. When EMS runs out, the storage manager gets the LRU EMS block. If that block is storage, the manager swaps it to disk and frees its EMS. If the block is a file block, the manager frees its EMS with no disk activity. Since a "clean" file block is already on disk, there is no reason to swap it out.

The block manager maintains an array of structures to keep track of blocks as they move between memory, EMS, and disk. This structure has the LRU lists and a pointer to where the block sits in memory, in EMS, or on disk. Each block handle corresponds to one array entry. The **block_preset** routine initializes this array during preset.

Four modules form the core of the block manager: BLOCK, EMS, SWAP, and FILE. BLOCK is the primary module. It controls the EMS, SWAP, and FILE modules, and it contains the application's interface routines. EMS and SWAP are very similar. They swap blocks to and from secondary storage. Because EMS is faster than disk, BLOCK favors it over SWAP.

THE INS AND OUTS

Both of the modules are of excellent design. They are simple and self-contained. With "_in" and "_out" routines in each, the interfaces are symmetrical and parallel. Their "_in" routines move a numbered block from secondary storage to memory, while their "_out" routines move a block from memory to secondary storage and return a block number.

A block-oriented interface to the DOS file system is implemented by way of the FILE module. Access to a file is prepared via **open_block_file**, while **read_block_file** transfers a block from

New Version 3.0

Commenting Disassembler!

SOURCER™ 486

- SEE HOW PROGRAMS WORK
- EASILY MODIFY PROGRAMS

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80386/387, 80486, and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

Sourcer is the best disassembler we've ever seen!
—PC Magazine, January 17, 1989, page 101

SAMPLE OUTPUT

Fully automatic

Program header

Assembler directives

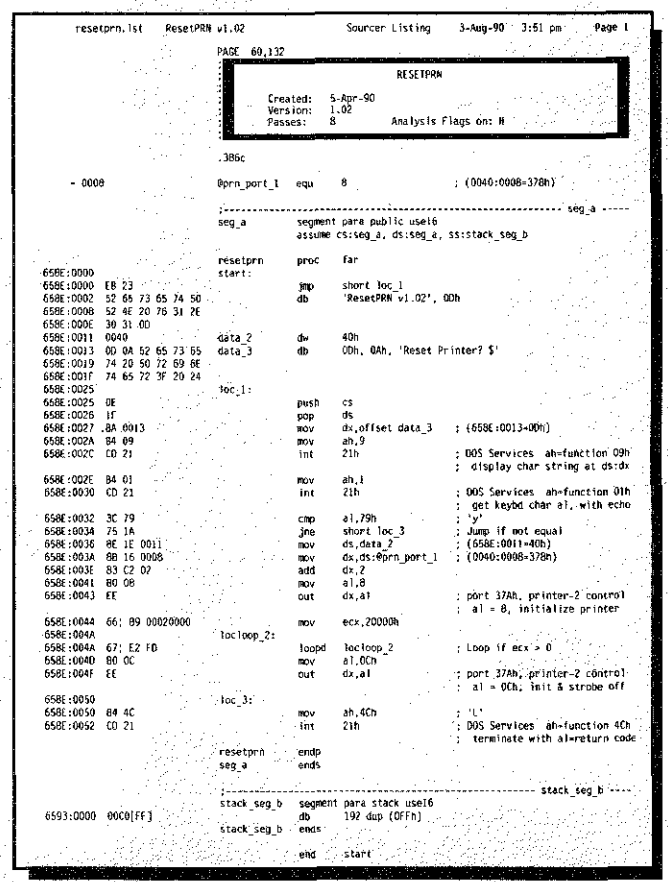
Determines data areas and type

Detailed comments

Simulator follows segment changes

80386 and 80486 support

Easy to read format


resetprn.lst ResetPRN v1.02 Sourcer Listing 3-Aug-90 3:51 pm Page 1
PAGE 60,132
RESETPRN
Created: 6-Apr-90
Version: 1.02
Passes: 8 Analysis Flags on: H
.386c
0prn_port_1 equ 8 ; (0040:0008=378h)
seg_a segment para public use16
assume cs:seg_a, ds:seg_a, ss:stack_seg_b
resetprn proc far
start:
658E:0000 EB 23 jmp short loc_1
658E:0002 52 65 73 65 74 50 db 'ResetPRN v1.02', 0Dh
658E:0008 52 4E 20 76 31 2E
658E:000E 30 31 0D
658E:0011 0040 data_2 dw 40h
658E:0013 0D 0A 52 65 73 65 data_3 db 0Dh, 0Ah, 'Reset Printer? $'
658E:0019 74 20 50 72 69 6E
658E:001F 74 65 72 3F 20 24
658E:0025
loc_1:
658E:0025 0E push cs
658E:0026 1F pop ds
658E:0027 BA 0013 mov dx,offset data_3 ; (658E:0013=0Dh)
658E:002A B4 09 mov ah,9
658E:002C CD 21 int 21h ; DOS Services ah=function 09h
; display char string at ds:dx
658E:002E B4 01 mov ah,1
658E:0030 CD 21 int 21h ; DOS Services ah=function 01h
; get keybd char al, with echo
658E:0032 3C 79 cmp al,79h ; 'y'
658E:0034 75 1A jne short loc_3 ; Jump if not equal
658E:0036 8E 1E 0011 mov ds,data_2 ; (658E:0011=40h)
658E:003A 8B 16 0008 mov dx,ds:0prn_port_1 ; (0040:0008=378h)
658E:003E 83 C2 02 add dx,2
658E:0041 B0 08 mov al,8
658E:0043 EE out dx,al ; port 37Ah, printer-2 control
; al = 8, initialize printer
658E:0044 66: B9 00020000 mov ecx,20000h
658E:004A
loc_loop_2:
658E:004A 67: E2 FD loopd loc_loop_2 ; Loop if ecx > 0
658E:004D B0 0C mov al,0Ch
658E:004F EE out dx,al ; port 37Ah, printer-2 control
; al = 0Ch, init & strobe off
loc_3:
658E:0050 B4 4C mov ah,4Ch ; 'L'
658E:0052 CD 21 int 21h ; DOS Services ah=function 4Ch
; terminate with al=return code
resetprn endp
seg_a ends
stack_seg_b segment para stack use16
6593:0000 00C0[FF] db 192 dup (0FFh)
stack_seg_b ends
end start

(Source code output and inline cross reference can also be selected)

BIOS SOURCE

- CHANGE AND ADD FEATURES
- CLARIFY INTERFACES

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$129.95	**ASMtool 486™** Assembly source analyzer and flowcharter	$199.95
SOURCER with BIOS Pre-Processor	169.95	**View-It™** High speed zoomable file viewer	69.95
UNPACKER™ Unpack packed EXE files	39.95	**ASM ProPak II™** Sourcer w/BIOS, UnPacker, ASMtool and View-It	399.75

Shipping & Handling: USA $5; Canada/Mexico $10; Other $15; CA Res. add sales tax; PS/2 trademark of IBM Corp.

NOT COPY PROTECTED **30-DAY MONEY-BACK GUARANTEE**

VISA **1-800-648-8266** MasterCard

V COMMUNICATIONS, INC.

4320 Stevens Creek Blvd., Suite 275, Dept. PJ8, San Jose, CA 95129 (408) 296-4224

Circle **125** on Reader Service Card

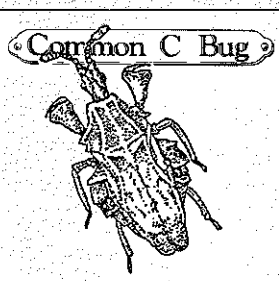
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

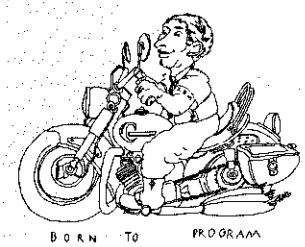
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$14.95 each — includes U.S. shipping



COMMON C BUG



BORN TO PROGRAM

PARALLEL PORTS

RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

PJ ISSUE SPECIAL! SAVE $2.00 ea.

ALL BACK ISSUES ONLY $5.00 EACH!

For a limited time, you can save over 28% on each back issue you order for your library. Complete your collection today -- order while supplies last!

Free U.S. shipping. Add $2.50 shipping per magazine to Canada or Mexico, $5.00 shipping per magazine to any other foreign destination. Offer expires 2-28-91.

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 7.4: COMMUNICATIONS: A simple terminal program in PM; communications and protocols; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual* and Programmer's Toolkit.

Vol. 7.5: OBJECT-ORIENTED PROGRAMMING: Class libraries in C++, from C to objects; improved C++ code generation; Hal Hardenbergh on the RISC/CISC schism; world scaling in Turbo Pascal's BGI graphics primitives; numeric extensions to C; reviews of 386 development tools, mini-Case tools, and Robert Sedgewick's book *Algorithms.*

Vol. 7.6: USER INTERFACES: Windows terminal program; the CURSES C library; two video displays with Turbo Pascal; VGA and EGA split screens; memory models and static variables for assembly language; object-oriented programming to create a file viewer; large memory spaces from DOS on the 80386; QuickBASIC's string-searching algorithm; reviews of user-interface and mini-Case tools; the *System BIOS for IBM PC, XT, AT.*

Vol. 8.1: NETWORKING: Network File System; an overview of NetBIOS on the PC, from *C Programmer's Guide to NetBIOS* by W. David Schwaderer; fast circle drawing on the VGA; Dynamic Data Exchange in Windows; DESQview API features; computing time in C; a menu system with Turbo Pascal 5.5.

Vol. 8.2: GRAPHICS: Faster circles for the VGA; the TIGA interface from TI; the EGA palette; 800x600x16 Super-VGA mode programming; dynamic data exchange for communications; Manipulating large data; fast file I/O in QuickBASIC; industry news; 1989 article index.

Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; graphics programming in windows; mousing around: messages in PM; accessing high memory; ASM:advanced segmentation; ellipses on the VGA; Up OOP alley in Pascal.

Vol. 8.4: PROGRAMMING IN THE PROTECTED MODE: Faster VGA ellipses and more; Running unprotected virtual-8086 mode; Converting to 386 protected mode; OS/2 2.0 memory model for the 386; FORTRAN for the 386; Using C++ features in C; Different times compensating for system clock idiosyncracies; ASM:Errors and stream I/O; Cobol-85 nested; Deliberately breaking the rules.

Vol. 8.5: OBJECT-ORIENTED PROGRAMMING: More colors on the VGA; A virtual memory system in C/C++; Pitfalls in Turbo Pascal OOP; Writing applications that travel; Late binding; Tools for 386 DOS; Windowing text with X; Cache memories; Breaking the rules again; SPARCs plot world takeover.

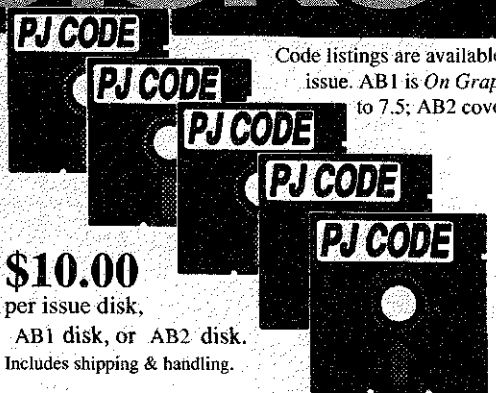
Vol. 8.6: THE DATABASE CHALLENGE: VGA color paging; The database dilemma; Networking databases; B-tree indexing; Virtual memory paging; Writing an ANSI C preprocessor; Peak performance: On to the 80486; Calendar conversions; Software components; Optimization strategies; Pink noise; Oh say can you C?

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00
per issue disk,
AB1 disk, or AB2 disk.
Includes shipping & handling.

Code listings are available from volume 5.5 to the current issue. AB1 is *On Graphics* series code from issues 5.1 to 7.5; AB2 covers issue 7.6 to the current issue.

Diskettes are available for the following code:

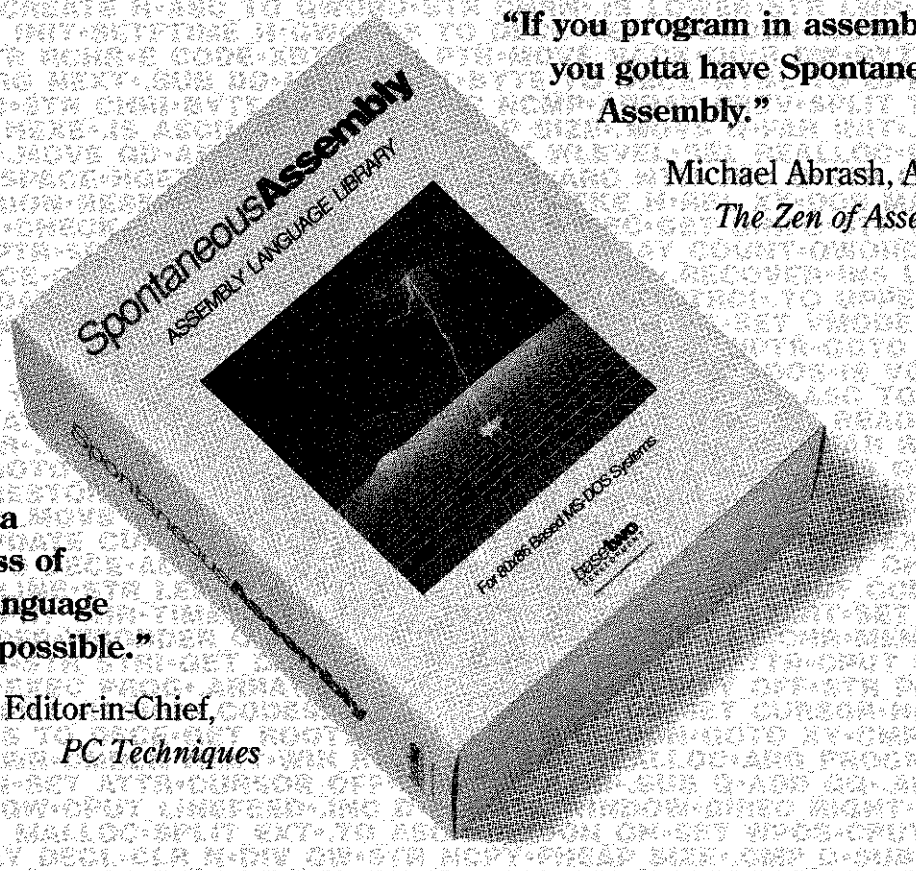
AB1	6.2	7.2	8.2
AB2	6.3	7.3	8.3
	6.4	7.4	8.4
5.5	6.5	7.5	8.5
5.6	6.6	7.6	8.6
6.1	7.1	8.1	9.1

360K MS-DOS disks standard; other formats are available.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"If you program in assembly language, you gotta have Spontaneous Assembly."

Michael Abrash, Author of
The Zen of Assembly



"Spontaneous Assembly makes a whole new class of assembly language applications possible."

Jeff Duntemann, Editor-in-Chief,
PC Techniques

Serious Assembly.

Nobody really takes assembly language seriously. Nobody, that is, except Lotus, WordPerfect, Novell, and everyone else who needs the tightest, fastest code possible.

But assembly is tough...unless you have plenty of ready-to-use assembly language routines at your fingertips. That's why industry leaders spend a lot of time and money creating their own, proprietary assembly language libraries.

You could spend your valuable development time building a library, or you can use **Spontaneous Assembly**, the complete **assembly language library** specifically designed for serious assembly language development.

Over 700 functions and macros.
• Complete text mode windowing system • Direct/BIOS/DOS screen I/O • Near/far/relative heap management • Program/environment control • Array management, sorting, searching • Critical error management • Quadword/dword integer math • Date/time manipulation • Enhanced DOS file I/O • File and directory management • String manipulation • Memory manipulation • Character/numeric/string conversion—and more. All carefully documented, tested, and ready to use.

Tight, fast code. Fast.
Every routine is hand-coded and hand-optimized (in assembly, of course). And a consistent, register-oriented parameter-passing convention makes these routines remarkably easy to use. With Spontaneous Assembly, coding in assembly is as fast as coding in a high-level language—without the overhead.

Powerful memory model support.
Spontaneous Assembly unleashes the full power of memory models in assembly language. It supports all Microsoft/Borland standard memory models as well as custom models and mixed-model programming. And it gives you complete control over segment/group names and attributes.

Complete documentation.
A 750 page reference manual describes every function, macro, and variable in detail. Step-by-step instructions and technical notes explain memory models, library customization, integration with C, and much more.

Spontaneous Assembly.
Serious assembly language programming doesn't need to be hard any more. Now it can be Spontaneous.

See for yourself. Order now from your dealer or call 1-800-ASSEMBLY to order direct. Then use Spontaneous Assembly for up to 30 days. If you're not 100% satisfied, return it for a full refund!

Retail Price.
$395 Plus shipping & handling. Includes 5 ¼" and 3 ½" diskettes.
Full source code included. No royalties.
30 day money-back guarantee.

Call Now!

1-800-ASSEMBLY

Orders • Information • Support

VISA • MasterCard • Check • C.O.D.

basetwo DEVELOPMENT

For 80x86–based systems running DOS 2.0 or later. MASM 5.1 or TASM recommended.
Base Two Development • A Division of Acclaim Technologies, Inc. • 11 East 200 North • Orem, Utah 84057 • (801) 222-9500
Spontaneous Assembly and Base Two Development are trademarks of Acclaim Technologies, Inc.

Circle 343 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        call    unlink_ems_list         ; delete from EMS LRU list
        mov     blk_ems_index[si],NULL_EMS
        jmp     brp4

brp2:   cmp     blk_file_id[si],NULL_ID ; check if file block
        je      brp3                    ;  if not

        push    bx                      ;  else read block from file
        mov     bx,blk_file_id[si]
        mov     ax,blk_file_index[si]
        call    read_block_file
        pop     bx
        jc      brp8                    ;  if error reading file
        mov     blk_byte_count[si],ax
        jmp     brp4                    ;  else read OK

brp3:   mov     ax,blk_swap_index[si]   ; check if block swapped out
        cmpx    ax,NULL_SWAP
        je      brp7                    ;  if not swapped-bad block handle
        call    swap_in
        jc      brp8                    ;  if error swapping block in
        mov     blk_swap_index[si],NULL_SWAP

brp4:   mov     wptr blk_pointer[si],di ; save RAM pointer in block table
        mov     wptr blk_pointer[si+2],es
        call    set_mru_ram             ; move to front of LRU list

brp5:   mov     ax,blk_byte_count[si]   ; return RAM pointer and byte count
        lds     si,blk_pointer[si]
        clc
brp6:   popm    es,di
        ret

brp7:   lea     ax,ertx_block_handle    ; *Bad block handle*
        call    set_strerror

brp8:   call    free
        stc
        jmp     brp6
block_read_primitive endp


;;      free ram block
;
;       exit    Cf      if no RAM free
;       uses    AX
;
free_ram_block proc
        call    save_most
        call    get_lru_ram             ; get least recently used memory
block
        jc      frb4                    ;  if no blocks on LRU list

        les     di,blk_pointer[si]      ; else file block, swap to EMS
        call    ems_out
        jc      frb2                    ;  if unable to swap to EMS
        mov     blk_ems_index[si],ax    ;  else save EMS page index
        call    set_mru_ems

frb1:   call    free                    ; free storage
        jc      frb4                    ;  if unexpected error
        mov     wptr blk_pointer[si],di ;  (free returns with ES:DI==NULL)
        mov     wptr blk_pointer[si+2],es
        jmp     frb4

frb2:   call    clear_strerror          ; here when EMS is full
        cmp     blk_file_id[si],NULL_ID ; is block file or storage?
        jne     frb1                    ;  if file block - free its RAM

        mov     bx,si                   ; try exchanging RAM block & LRU
EMS
        call    get_lru_ems
        jc      frb3                    ;  if no EMS (SI still-> RAM block)
        mov     ax,blk_ems_index[si]
        call    ems_exchange            ;  (ES:DI still-> RAM block)
        jc      frb4                    ;  if exchange failed (serious error)

        movx    ax,NULL_EMS             ; exchange EMS index & RAM pointers
        xchg    ax,blk_ems_index[si]    ;  for exchanged blocks
        mov     blk_ems_index[bx],ax
        mov     wptr blk_pointer[bx],NULL_POINTER
        mov     wptr blk_pointer[bx+2],NULL_POINTER
        mov     wptr blk_pointer[si],di
        mov     wptr blk_pointer[si+2],es
        xchg    bx,si                   ; set most recently used EMS block
        call    set_mru_ems
        xchg    bx,si

frb3:   call    swap_out                ; write block to swap file
        mov     blk_swap_index[si],ax   ;  set swap index (or 0 if failure)
        jnc     frb1                    ;  if swapped OK

        call    set_lru_ram             ; swap failed, re-link RAM block
        movx    bx,NULL_BLOCK
        stc

frb4:   ret
free_ram_block endp


;;      get lru ems
;
;       exit    DS:SI   least recently used block in EMS
;               Cf      if no blocks in EMS (SI unchanged)
;       uses    AX
;
get_lru_ems proc
        mov     ax,OPEN_LINK
        cmp     ax,ems_back[bp]
        je      gle2                    ; if no EMS blocks

        lds     si,block_table[bp]
        mov     si,ems_back[bp]
        cmp     si,ems_front[bp]
        je      gle1                    ;  if only one EMS block

        push    bx
        mov     bx,blk_forward[si]
        mov     blk_backward[bx],ax
        mov     ems_back[bp],bx
        pop     bx
        clc
        ret

gle1:   mov     ems_front[bp],ax
        mov     ems_back[bp],ax
        ret                             ;  (Cf==0)

gle2:   stc
        ret
get_lru_ems endp


;;      get lru ram
;
;       exit    DS:SI   least recently used block in RAM
;               Cf      if no unlocked blocks in RAM
;       uses    AX
;
get_lru_ram proc
        mov     ax,OPEN_LINK
        lds     si,block_table[bp]
        mov     si,ram_back[bp]
        cmp     si,ax
        je      glr2                    ;  if no RAM blocks
        cmp     si,ram_front[bp]
        je      glr1                    ;  if only one RAM block

        push    bx
        mov     bx,blk_forward[si]
        mov     blk_backward[bx],ax
        mov     ram_back[bp],bx
        pop     bx
        clc
        ret

glr1:   mov     ram_front[bp],ax
        mov     ram_back[bp],ax
        ret                             ;  (Cf==0)

glr2:   stc
        ret
get_lru_ram endp


;;      malloc storage block
;
;       exit    ES:DI   storage block
;               Cf      if unable to allocate storage
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

disk to memory. When a file opens, FILE checks to see if the file was previously open. If it was, the program can save slow DOS calls should all or part of the file still sit in RAM or EMS. FILE has two limitations, though: it doesn't detect when a file changes between opens, and its only access mode is read.

TEST_BLK is a test program for the block manager. It allocates 100 storage blocks (1.6 MB), selects and reads a random block 1,000 times, opens a file from the command line, reads the file blocks in reverse order, and then reads 100 more random blocks.

INITIALIZERS/TERMINATORS

In a previous column on segments, I discussed a technique for building link-time tables. In this column, I will develop this technique for the standard start-up code.

Three link-time tables contain lists of addresses for initializers and terminators. The start-up code calls each address in the initializer table before calling **main**. When **main** returns, the start-up code calls each address in the full exit table and in the quick exit table. Each table requires three segments. XIB, XI, and XIE hold the initializer table. XPB, XP, and XPE hold the pre-terminator table. XCB, XC, and XCE hold the quick exit table. The same segment names appear in the start-up code for Microsoft C.

Even though two of the three segments are empty, always define all three in modules. For example, always define XIB before XI and XIE after XI. If you do not, the segments can rearrange themselves and your program will fail.

The quick exit table in XCB, XC, and XCE handles emergency program exits. Table entries do functions such as freeing EMS when a program aborts. Before adding entries, be sure to read and understand the following section.

EXIT INTERCEPT

Programs using EMS and swap files must have special code to free EMS and delete a swap file when the program terminates. The design of this code is important because programs can terminate in unexpected ways. The worst-case examples of termination happen during critical DOS errors and debugging. The only guaranteed way to trap all possible program terminations is to hook the **terminate** address at 0x000A in the program segment prefix (PSP). DOS returns control to this address after a program terminates.

The **terminate** address is dangerous to use because, at that point, DOS thinks the program is gone, all files are closed, and all memory blocks are free. The active PSP is COMMAND.COM. But the program is still in memory and can execute instructions, *if* it is very careful. Careful means avoiding DOS calls.

Unfortunately, the block manager must call DOS in order to delete the swap file. Note that the block manager cannot close the swap file during exit intercept, because it is already closed. The code to hook the **terminate** address is in TERM.ASM. Notice that the code only hooks the address on request via **enable_exit_intercept**. On a full exit, TERM automatically disables the exit intercept.

NEW MACROS

The ASM.INC **include** file has some new macros. They permit a variety of useful optimizations. MOVX moves between segment registers with a **push** and a **pop**. It also zeros a register with a **sub** instruction.

```
movx  ds,DGROUP_SEGMENT  ; this

push  ss                 ; instead of this
pop   ss

movx  bx,NULL_POINTER    ; this

sub   bx,bx              ; instead of this
```

Use CMPX to compare a register to zero. It saves a byte by substituting a two-byte **OR** instruction for CMP.

```
cmpx bx,NULL_POINTER  ; expands to OR BX,BX
```

ZER0 is an equate for zero. Since *bp* is always zero in my assembly language programs, I can change this equate and save one or two bytes in each of hundreds of instructions. In the example shown below, you can save two bytes if **ZER0==BP**. Note that the last character of **ZER0** is a digit.

```
mov  ems_counter[bp],ZER0
```

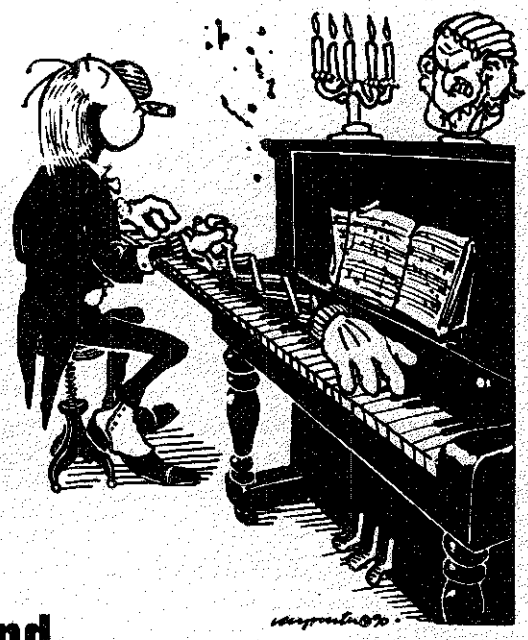
If you missed my previous recommendations against macros, they still apply. Avoid macros if possible. If you must use them, make their functions trivial and obvious.

```
;       uses    AX
;
malloc_storage_block proc
        push    cx
        mov     cx,BLOCK_SIZE           ; allocate block of memory
        call    malloc
        jnc     msb1                    ;  if OK

        call    clear_strerror          ; clear malloc error
        call    free_ram_block          ; swap to EMS or disk to free RAM
        jc      msb1                    ;  if too many locked blocks

        call    malloc                  ; try malloc one more time

msb1:   pop     cx
        ret
malloc_storage_block endp


;;      new block handle
;
;       exit    BX      block handle
;               DS:SI   block table entry
;               Cf      if no free handles
;       uses    AX
```

```
new_block_handle proc
        lds     si,block_table[bp]      ; get first block from free list
        mov     si,free_front[bp]
        cmp     si,OPEN_LINK
        je      nbh1                    ;  if no more handles
        mov     ax,blk_backward[si]
        mov     free_front[bp],ax
        dec     free_count[bp]

        push    dx                      ; compute handle from table offset
        mov     ax,si
        movx    dx,0
        mov     bx,size block_str
        div     bx
        add     ax,FIRST_BLOCK
        mov     bx,ax
        pop     dx
        ret

nbh1:   lea     ax,ertx_virtual_full    ; *Virtual memory full*
        jmp     set_strerror
new_block_handler endp

; All the listings are on the PJ 9.1 code disk, and some will be printed
; in the next issue
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Enhanced keyboards



Writing for the extended keyboard

By Mike Schmit

You've probably never thought much about your keyboard, and with good reason—you press a key, something shows up on the screen, and that's about all there is to it. However, some programs may require a little more. For example, TSRs need to detect various combinations of keys that are not normally available. Application programs might want to display different on-screen options when the Control or Alt keys are down. Clearly, the dumb terminal type of keyboard interface is not sufficient for many programs.

Advanced keyboard programming is more than just learning how to read the F11 and F12 keys. We'll start by describing the flow of data from the keyboard to a program after a brief note on terminology. The term "scan code" is used here to specify the code that is passed from the keyboard to the CPU, while "key code" will mean a code used internally by the CPU (which may be the same as or one of a superset of the scan codes).

KEYSTROKE DATA PATH

When a key is pressed (or released) the keyboard controller chip generates a hardware interrupt, which is directed to interrupt 9 under DOS. The interrupt service routine (ISR) for INT 9 reads the scan code from the hardware and converts the scan code to a key code and ASCII code (if any) based on the state of the shift status (see Table 1). The shift status consists of the Shift, Control, Alt, Caps Lock, and Num Lock keys. The BIOS routines ignore most combinations of shift states (except, for example CTRL-ALT-DEL). For all other purposes there is an order of precedence in checking the shift status bits. The order is Alt, Control, then Shift. The Num Lock state inverts the shift state for the key pad, and the Caps Lock state inverts the shift state for alphabetic characters.

Next the key code and ASCII code are placed into the keyboard buffer (also called the type-ahead buffer). The key then waits in the buffer until a program requests to read a key via one of the BIOS INT 16h functions. The various DOS functions (such as INT 21h functions 01, 06, 07, 08, 0Ah, 0Bh, and 0Ch) use INT 16h internally.

A CAPSULE HISTORY

The keystroke-to-BIOS model that is described above is fairly easy to understand and was all you needed to know to program on the original PC with the 83-key keyboard. However, when the PC AT came out, several changes were made. A compatible 84-key keyboard was introduced (adding the SYS REQ key), followed by the 101-key keyboard (102 keys in the foreign version). The SYS REQ key was apparently designed into the BIOS for "multitasking" support, but cannot be read by INT 16h (except that the state is returned by function 12h) and is rarely found in use by any program. The 101-key keyboard, with 12 function keys across the top and some duplicate editing keys, was called "enhanced" and of course the BIOS added "enhanced" support.

Since many programs had been written that were not expecting any of these new keys, a program might crash if an unexpected key was read by the program. Therefore, the smart thing to do was to change the INT 16h functions (0,1,2) as shown in Tables 1 and 2. This way, all of the new keys would be ignored, and the codes sent by duplicate keys would be converted to seem the same as the original keys. New functions—INT 16h functions 10h, 11h, and 12h—were added to the BIOS to allow reading of the new codes. To be able to read any of the new keys or to tell the difference between one of the original editing keys and the new duplicate ones, these new functions must be used.

F11 AND F12 SUPPORT

Adding support for the F11 and F12 keys is the primary reason for most applications to care about the enhanced

Mike Schmit is president of Quantasm Corporation. His specialty is assembly language programming. Contact Mike at 19855 Stevens Creek Blvd., Suite 154, Cupertino, CA 95014; (404) 244-6826.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 1: BIOS keyboard services

```
AH
0  Read Character
     returns: AL=ASCII code; AH=key code
1  Get Keyboard Status
     returns: Zero flag set if no key,
              else same as function 0
2  Get Keyboard Flags
     outputs: AL=keyboard flags (see below)
3  Set Repeat Rate(*)
     inputs:  AL=5 (0-4 used on PCjr);
              BL=repeat rate (00-1fh) (30 to 2 chars/sec);
              BH=repeat delay (0-3) (250,500,750,1000 msec)
4  Set Keyclick (PCjr only)
5  Push Character & scan code(*)
     inputs: CL=ASCII code; CH=key code
     outputs: Carry flag set if buffer is full
10 Read Enhanced Character(*)
     outputs: same as function 0
11 Get Enhanced Keyboard Status(*)
     outputs: same as function 0
12 Get Enhanced Keyboard Flags(*)
     outputs: AX enhanced keyboard flags (see below)
```

KEYBOARD FLAGS

(bits 0-7 returned by function 02h)
(bits 0-15 returned by function 12h)

Bit	Description (if set)
0	Right shift key down
1	Left shift key down
2	Either control key down
3	Either Alt key down
4	Scroll Lock toggle on
5	Num Lock toggle on
6	Caps Lock toggle on
7	Insert toggle on
8	Left Ctrl key down
9	Left Alt key down
10	Right Ctrl key down
11	Right Alt key down
12	Scroll key down
13	Num Lock key down
14	Caps Lock key down
15	Sys Req key down

* = only available on newer BIOSes

keyboard. This is not as simple as you might assume. Support for the new functions in the BIOS does not mean that a new keyboard has been installed, though lack of support does imply an old keyboard.

The easy way to know if the BIOS on your machine supports the new "enhanced" functions is to check for a 101-key keyboard; if one has been installed, the BIOS must support the new calls. If it doesn't have the keyboard, assume that it doesn't have the support. This works for most purposes, except that you may want to use some of the other BIOS features, such as setting the key repeat rate (see Table 1), even if a 101-key keyboard has not been installed.

CHECKING THE BIOS

An algorithm that I have found to be reliable is as follows:

1) Flush the keyboard buffer by making calls to INT 16h function 1, then to function 0 if a key is available. Repeat until no keys are available.

2) Load a dummy key code (such as 0FFFFh) into the keyboard buffer using INT 16h function 5. Since this service is available only on the new BIOS, if it fails, the test is over and an old BIOS has been found.

3) Test to see if the dummy key code can be read using INT 16h function 10h. Full support is available if this key code can be read.

Detecting the type of keyboard installed is much easier. The BIOS stores the keyboard type in bit 4 (10h) of the status byte at the address 40:96h. A 1 bit indicates that a 101-key keyboard has been installed.

This is all that you need to know for most programs, and you can now allow the use of the F11 and F12 keys. For most purposes, it is rarely necessary (and may even be considered poor programming style) to differentiate between the duplicate keys, such as the new home key and the home key on the 7 on the key pad. Programs such as TSRs or task switchers, however, may specifically choose to differentiate between these keys for their own special purposes.

READING THE KEYS

So now your program has detected that a 101-key keyboard has been installed, and you call INT 16h function 10h to read a key. For compatibility with previous key codes while including the

TABLE 2: Return codes (AH/AL) for INT 16h functions

Keys available only on 101/102-key keyboards

		INT 16h FUNCTION 00h/01h			INT 16h FUNCTION 10h/11h			
KEY	SCANCODE	NORMAL	SHIFT	CONTROL	NORMAL	SHIFT	CONTROL	ALT
Pad Enter	E0-1C	1C/0D	1C/0D	1C/0A	E0/0D	E0/0D	E0/0A	A6/00
Right Ctrl	E0-1D							
Pad /	E0-35	35/2F	35/2F		E0/2F	E0/2F	95/00	A4/00
Right Alt	E0-38							
Home	E0-47	47/00	47/00	77/00	47/E0	47/E0	77/E0	97/00
Up	E0-48	48/00	48/00		48/E0	48/E0	80/E0	98/00
Page Up	E0-49	49/00	49/00	84/00	49/E0	49/E0	84/E0	99/00
Left	E0-4B	4B/00	4B/00	73/00	4B/E0	4B/E0	73/E0	9B/00
Right	E0-4D	4D/00	4D/00	74/00	4D/E0	4D/E0	74/E0	9D/00
End	E0-4F	4F/00	4F/00	75/00	4F/E0	4F/E0	75/E0	9F/00
Down	E0-50	50/00	50/00		50/E0	50/E0	91/E0	A0/00
Page Dn	E0-51	51/00	51/00	76/00	51/E0	51/E0	76/E0	A1/00
Ins	E0-52	52/00	52/00		52/E0	52/E0	92/E0	A2/00
Del	E0-53	53/00	53/00		53/E0	53/E0	93/E0	A3/00
|	56	56/5C	56/7C		56/5C	56/7C		
F11	57				85/00	87/00	89/00	8B/00
F12	58				86/00	88/00	8A/00	8C/00

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

new F11 and F12 codes, the following algorithm will do the trick:

```
IF AX = 00E0h then
        (user entered 224 on keypad)
ELSEIF AL = E0h then
        (found one of ten duplicate editing keys)
ELSEIF AH = E0h then
        (found keypad ENTER or forward slash)
ELSE
        (found old key, F11, F12, or new Ctrl or
          Alt combination)
ENDIF
```

Notice that it is not perfectly simple. For example, it is rare that a user would want (or be allowed) to enter ALT 224 from the keypad (it produces the Greek alpha symbol), but most programming editors do allow every graphic symbol to be entered. If AL = E0h, then you have found one of the duplicate editing keys, seen in Table 3. Next, if AH = E0h, you have found one of the two duplicate keys that have ASCII codes—the key pad's ENTER key or its forward slash (divide) key. Most applications do not need to check for this, since the ASCII code is all they use. Finally, if none of the above conditions have been met, just use the AL and AH values as before. AL is the ASCII code of the key unless it is zero, whereupon there is no ASCII code and the key code is in AH. (Other ways of describing the full algorithm are shown in Figure 1.)

HIGH-LEVEL INTERFACE

So far, so good. Now you want to do all this from a high-level language like C, BASIC, or Pascal. And you want the code to be portable. You'd like the same code to run on a VAX running VMS and on a SUN running UNIX. The solution is simple—because the machines are different, you need separate keyboard entry routines for each machine, unless you want a program that can only read ASCII characters from the keyboard. The version for one or more of the operating systems will have to convert keyboard codes to the codes of your primary machine; if it doesn't, you will have to make up a set of "virtual" codes.

FIGURE 1

```
key code algorithm:
IF AX = 00E0h then
        (user entered 224 on keypad)
ELSEIF AL = 00h then
        (key code in AH)
ELSEIF AL = E0h then
        (AH has key code for one of
         ten duplicate editing keys)
ELSE
        AL is ASCII code  (IF AH=E0
        found keypad ENTER or slash)
ENDIF
```

The DOS interface to the keyboard (i.e., INT 21h functions 7 and 8) will return a 0 if there is no ASCII code, then return the key code the next time the function is called. The DOS handle function (3Fh) does not accept any non-ASCII keys. DOS 2.x and 3.x seem to use only INT 16h functions 0 and 1, while DOS 4.x checks and uses function 10h and 11h. So if you use the DOS calls you cannot use the F11 and F12 keys (unless you require DOS 4.x to run your program). Use the BIOS functions to read the keyboard through whatever mechanism you have in a high-level language, such as a function that passes a register structure to INT 16h.

Another keyboard feature you can use (primarily to detect hot keys in TSRs) is the way that the INT 16h function 12h returns bits set for each of the shift keys. The 101-key keyboards have two Alt keys and two control keys, in addition to the standard two shift keys. INT 16h function 12h returns two status bytes, as shown in Table 1.

KEYBOARD INTERCEPT

With enhanced keyboard support came tremendous support for hot key detection in TSRs. INT 15h function 4Fh, the keyboard intercept, is probably one of the least known and most poorly documented of the new BIOS routines. After booting, the standard BIOS routine does nothing but an IRET. The INT 9 ISR calls this function each time a key is pressed or released.

The input to this routine is AL (scan code). When a key is being released, the high bit in AL is set. Your routine can then change the scan code. The carry flag should be cleared if the key is to be ignored. If INT 15h is intercepted, control should be passed on to the previous INT 15h, just as with any other interrupt. In addition, INT 9 generates some special pairs of internal codes—FAh-FFh—for certain keys such as Caps Lock, Num Lock, and Scroll Lock. Keys that are duplicates on the 101-key keyboards send an E0h first, followed by the same scan code as that assigned on previous keyboards. Sample code for the INT 15h handler is in Listing 1.

LISTING 1: Sample INT 15h intercept

```
New_int15 Proc Far
    pushf
    cmp   ah, 4fh       ; keyboard intercept?
    je    new15_1       ; jump if so
    popf
    jmp   CS:old_int15  ; pass on
new15_1:
    popf
    sti                 ; flags on stack
                        ; & INT Flag set
    pushf
    cli
    call  CS:old_int15  ; pass interrupt
    jnc   new15_2       ; exit if other program
                        ; wants key ignored
; put your own test code here

new15_2:
    ret   2             ; return from interrupt
                        ; discard saved flags
New_int15 Endp
```

MORE THAN YOU KNOW

If you want to learn more about the way that your keyboard operates, you have several options. For learning about INT 16h functions, you can write an assembly language program or a high-level language program that invokes the various INT 16h functions. Inspecting the operation of INT 9 and INT 15h is a bit more difficult; the best way is with a debugger that allows remote debugging via a serial port. Alternatively, you can write a program that monitors the keystrokes, or a filtered subset of them, and records them in a buffer. Examining the buffer should give you a sense of what to expect from the different keystrokes on a given keyboard.

As you see, there is more to know about the keyboard than you probably cared to know. With any luck, this article has supplied you with the information you need to write applications that have the features you want and that work on most keyboards. PJ

TABLE 3: 101-key keyboard changes

Duplicate keys	New keys
Home	up arrow	F11
End	down arrow	F12
Ins	left arrow	
Del	right arrow	
Pg Up	Enter (on keypad)	
Pg Dn	/ (on keypad)	

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For a Few Good Books

By M. Steven Baker

This is the first of what will be a regular column about books for programmers. Though I'm in Las Vegas tonight with Fall Comdex in full force, I would prefer to be sitting in a comfortable armchair at home reading a good novel—but you can't have everything. My comments in this space will not be full-blown, conventional book reviews. I think a different formula is in order. Every month the computer press churns out an enormous number of titles targeted for the entire spectrum of computer programmers. A few reviews every issue wouldn't even skim the surface. So I'll mention a number of recent and semi-recent books, grouped here by topic, that I consider worthwhile additions to your library.

DOS AND OPERATING SYSTEMS

One book stands out as the best all-around reference for DOS—Que Press's ***DOS Programmer's Reference*** by Terry Dettman. The second edition came out in 1989 and covers DOS up through version 4.01 (updated by Jim Kyle). The first half of the book covers DOS and the PC ROM BIOS. Chapters describe the structure of DOS, DOS/BIOS interface, serial devices, disk internals, directories, files, program and memory management, interrupt handlers, and device drivers. These sections use examples in C, BASIC, and Pascal to illustrate the typical DOS interface issues.

The second half is a complete reference to INT functions (hardware, ROM, and DOS) up through INT 2Fh, with mouse and EMS functions also getting good coverage. Due to space limitations, a few items like NetBIOS are not covered. Although Ray Duncan's book from Microsoft Press is good as a reference, Terry Dettman's title goes further. For example, all undocumented DOS calls ("reserved" functions in Microsoft jargon) and some DOS internals are covered. Information on undocumented DOS calls is essential in many situations, such as programming TSRs or low-level disk utilities. Dettman and Kyle have succeeded in providing a complete reference.

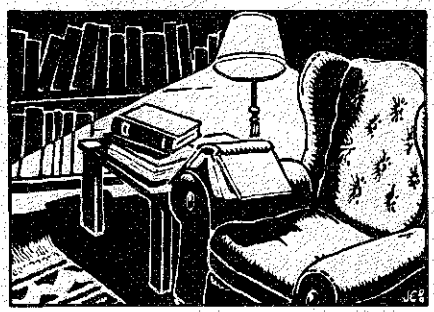


ILLUSTRATION BY JIM CARPENTER

If your interest is in more sophisticated operating system issues, particularly networking, I recommend ***The Design and Implementation of the 4.3BSD UNIX Operating System*** by Leffler, McKusick, Karels, and Quarterman from Addison-Wesley. The authors were major players at the University of California at Berkeley during the development of BSD flavors of UNIX. Innovations from this group include job control, demand-paged virtual memory, interprocess-communication (IPC), and improved networking with TCP/IP and the BSD socket interface.

This book focuses on facilities, data structures, and algorithms used in implementing 4.3BSD features. The first part covers history, design goals, and kernel services. Both process and memory management are given detailed discussion. Descriptions are provided for the I/O system, including the file system, device drivers, and terminal handling. I found the last section on interprocess communication and networking (sockets, network interfaces, and protocols) to be the most valuable. If you want to understand how a layered protocol can be implemented efficiently with sample data structures and storage management, you definitely should check out this book.

If you need software references to the latest Intel chips, Osborne McGraw-Hill is reprinting the Intel manuals. Titles include ***i486 Microprocessor Programmer's Reference Manual***, ***386SX Microprocessor Programmer's Reference Manual***, and the ***i860 Microprocessor Architecture***, the latter by Neal Margulis. The Osborne reprints are often priced a dollar less than Intel's manuals; look for them at your local bookstore.

GRAPHICS AND WINDOWS

Brady has a good new reference-quality book on low-level VGA graphics—***Advanced Programmer's Guide to SuperVGAs*** by George Sutty and Steve Blair. This title is a follow-up to the authors' 1988 ***Advanced Programmer's Guide to the EGA/VGA***. The first section contains a review of the VGA, but the real substance of the book is the SuperVGA programming detailed in sections two and three. This is the only place I've found reference details on a number of SuperVGA boards. A general section discusses the range of typical architectures found in SuperVGA boards, including mapping of display memory, memory paging, and enhanced display modes. A more detailed section describes how to program examples in assembly language that will read/write pixels, draw lines, fill rectangles, copy blocks, and handle cursors. These examples are then repeated in separate sections for 4-color, 16-color, and 256-color modes.

Separate chapters cover the specifics of individual chipsets, including Ahead, ATI, Cirrus, Chips and Technologies (BOCA), Genoa, Headland (formerly Video 7), Trident (Everex), Tseng (Orchid), Western Digital, and Zymos. For each chipset, information on new display modes, memory organization, new registers, chipset detection and identification, and display memory paging is given, along with programming examples.

Unique features of each chipset (e.g., zooming and hardware cursors) are illustrated, and the VESA standard is covered as well. Some tables differ from the text, which indicates a need for better proofing. The programming examples are enclosed on a 1.2 MB floppy diskette in the back cover.

Windows 3.0 programming books are starting to roll off the press. The second edition from Microsoft Press of ***Programming Windows*** by Charles Petzold is an excellent reference. Petzold introduces enhancements in Windows 3, such as virtual memory, multitasking, Dynamic

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Data Exchange, and the Multiple Document Interface. This remains the classic Windows programming guide.

I've borrowed a copy from the Comdex show floor of Bantam's ***Windows 3.0 Power Programming Techniques*** by Peter Norton and Paul Yao. I've only had a chance to read part of this, but Paul Yao comes highly recommended by Frank Greco. Yao authored an earlier Windows book that was published by Brady in 1987. This new book appears very readable and uses the Microsoft Hungarian notation similar to the Windows SDK. The focus is more on graphics programming than on text applications. The reader is led through the Windows environment and given plenty of example code.

DATABASE PROGRAMMING

If any one author stands out in my database library, it is C. J. Date. A purchase of any title by this talented writer is a good investment. A couple of recent books may be of particular interest. The second edition of Date's ***A Guide to the SQL Standard*** was published by Addison-Wesley in 1989. This book is an excellent reference on ANSI SQL, emphasizing the use of SQL for "programmed" (as opposed to "interactive") access to databases (embedded SQL). Date's *Guide* covers differences between the IBM and ANSI SQL versions and examines proposed extensions to SQL2, the X/OPEN SQL dialect (common under UNIX), and the latest IBM DB2 SQL dialect.

Date also wrote ***Relational Database Writings 1985-1989*** from Addison-Wesley in 1990, a collection of papers on relational database technology. Besides Date's own essays, this book features a number of Andrew Warden's "Adventures in Relationland" which discuss batched updates, column naming, natural joins, and primary key inheritance. The large section on relational database management has articles on domains, defining data types, referential integrity and keys, problems with NULLS, relational myths, and distributed databases. A major section on SQL offers critiques of logical flaws, do's and don'ts, dates and times in IBM SQL, and what's wrong with SQL.

In 1989, Microtrend Books published ***The dBASE Language Handbook*** by David Kalman, editor of *Databased Advisor* magazine. If you do much cross-tool database development with dBASE compatibles, this book is for you. This reference work is an alphabetized list and description of commands and functions supported by Quicksilver, Clipper (summer '87), dBXL, dBASE III Plus, dBASE IV (version 1.0), and FoxBASE+. Specific notes detail differences between keywords in different environments. The entries also reference other related functions or commands. As a dBASE-compatible language reference, this book is an excellent choice. I'm hoping a second edition is in the works.

Total control with LMI FORTH™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

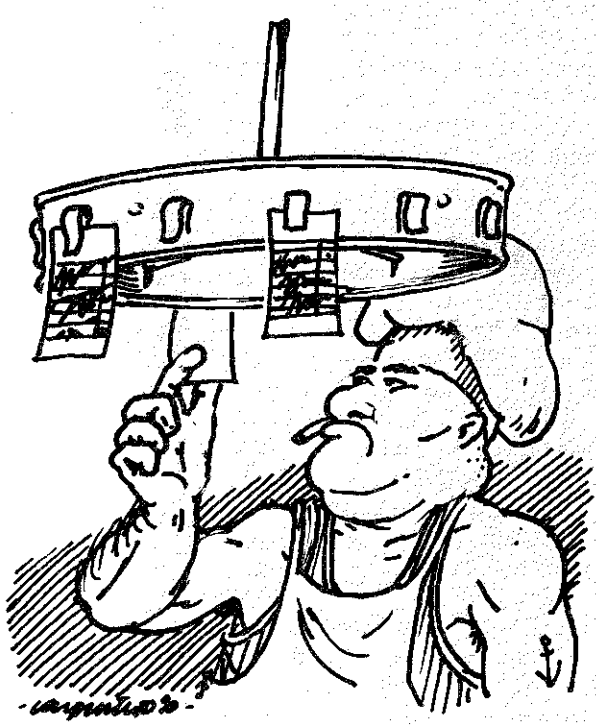
For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PRESSROOM

TURBO PASCAL 6.0 WITH OOP

Borland is now shipping version 6.0 of Turbo Pascal, its object-oriented programming language. New features include Turbo Vision, an OOP application framework enabling programmers to create applications with built-in support for overlapping windows, pull-down menus, and mouse/keyboard event handling. An Integrated Development Environment (IDE) offers overlapping windows and mouse support, a multi-file editor, dialog boxes, cut-and-paste clipboard, and integrated debugging with a new register window and conditional breakpoints. Additional features of 6.0 include an online tutorial and a hypertext help system. A built-in assembler gives programmers access to Pascal symbols and allows the addition of assembly language routines.

Turbo Pascal Professional 6.0 adds the Turbo Drive compiler, which uses extended memory; Turbo Debugger, which supports object-oriented debugging and reverse execution, and debugs programs of any size on 286 or 386 PCs with extended memory; an interactive profiler; and the Turbo Assembler, which supports 486 instructions.

Turbo Pascal 6.0 is $149.95, and Turbo Pascal Professional 6.0 is $299.95. Upgrades are also available.

Borland International
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95066-0001
(408) 439-1880

LEARNING TO DO WINDOWS

WinSoft has released Instant Windows, a data-validating screen library for Windows 3.0 and DOS. Designers can use the Instant Windows prototyper to design and animate menus and to enhance control flow and data entry in windows. Programmers can add their own internals, database, and communications function calls to the C code from the code generator.

Data field validation includes built-in data type/range checks, picture and display-only fields, passwords, and other controls. Data types include short and long integer, floating point, graphical and right-justified numbers, money, date, time, and text. Programmers can also attach custom validation functions to fields for screen data lookups and custom or cross-field validations.

An object-oriented screen engine manages all GUI objects, including windows, dialog boxes, prompts, tables, menus, radio buttons, check boxes, icons, scroll bars, text files, colors, and messages. Scrolling and zooming-to-form tables allow users to edit, display, add, move, copy, or delete rows of data.

Porting between DOS and Windows is done in minutes simply by relinking. Instant Windows requires Microsoft C 5.1 and DOS 3.0 or higher. The Windows 3.0 version requires the Windows SDK. There are no royalties or runtime fees. The DOS version sells for $249, and the Windows 3.0 version is $495.

WinSoft
1016 E. El Camino Real #216
Sunnyvale, CA 94087
(415) 324-9552

NEW MICROSOFT TOOLS

Microsoft is offering two new programming tools: the Microsoft Professional Advisor Library and the Microsoft C Developers Toolkit. A maintenance upgrade is also available to Microsoft C 6.0.

The Professional Advisor Library allows application developers to add hypertext-based online help systems to DOS and OS/2 applications. Routines are provided for basic file management, topic look-up, context maneuvering, support for text attributes, and utility routines for handling multiple help files. There are no royalty fees.

The C Developers Toolkit has software and documentation to help developers create tools and utilities that will integrate into Microsoft Programmer's WorkBench. The toolkit includes documentation on CodeView records and on the Source Browser API library to let users query a project database with the same browsing capabilities as those of Programmer's WorkBench. The Toolkit facilitates the development of products that work seamlessly from within Professional Development Systems. The Microsoft C 6.00a upgrade includes an improved compiler, CodeView 3.11, with mixed-language support, VCPI compatibility, and unlimited nested include files in Programmer's WorkBench 1.10.

The Professional Advisor Library is $49.95, and the C Developer's Toolkit is $29.95. The C Professional Development System 6.00a upgrade is $12.95.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
(800) 426-9400

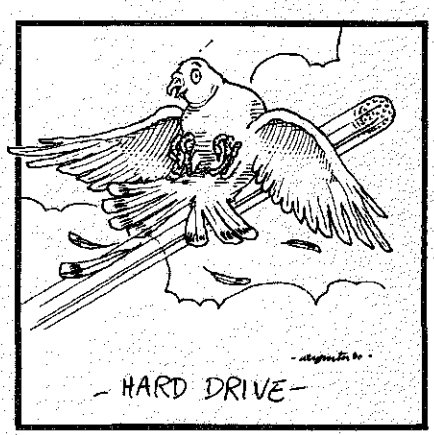
NEW FROM METAWARE

MetaWare has released two products: Globally Optimizing High C Compiler 2.3 for extended DOS 386/486, and 32-Bit Source-Level Debugger for use with MetaWare's High C compiler.

The compiler includes global optimizations and a new function inliner to enhance the performance of generated code. OS/2 2.0 32-bit protected mode is supported, as is Phar Lap's 386|DOS Extender. The code generator uses Weitek and Intel math coprocessor instruction sets, including support for in-line transcendentals and floating point long doubles (80 bits). Compiled modules can be linked with modules from MetaWare's Professional Pascal and modules from other C, Pascal, Ada, and FORTRAN compilers.

The High C compiler supports MS-DOS, OS/2, UNIX, Sun, and other platforms. Programs created with the High C compiler must be run under Phar Lap's or Ergo's 386 DOS extender.

MetaWare's 32-Bit Source-Level Debugger allows software developers to access virtually any size program by module name, line number, or text pattern. Breakpoints can be set for the value of a C integer expression and can execute a macro or a command. The user interface includes on-line help, overlapping and resizeable windows, scroll bars, pop-up and pull-down menus, source windows, control windows, and dialog



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ Code

is available on the following bulletin board services:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from Sept/Oct 1987 issue through the current issue.

A.C.M. Phoenix Chapter
P.O. Box 47302
Phoenix, AZ 85068
(Data) 602-970-0474

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Lynzie's Motherboard
3960 Laurel Canyon Blvd., Ste. 121
Studio City, CA 91614
(Voice) 213-654-0526
(Data) 213-650-0519 or 650-1233
Type PJ after your name at LOGON for access within 24 hours without verification.

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention PJ when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

The Programmer's Corner
5430F Lynx Ln. #224
Columbia, MD 21044
(Data) 301-596-1180, 621-4636, or 997-3744

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33441
(Voice) 305-360-9362
(Data) 305-360-9376

boxes. Data, registers, and breakpoints can be edited through the windows.

The debugger requires High C 2.3, Phar Lap 386|DOS Extender, a linker, and a 386/486 with at least 2MB of RAM. The Globally Optimizing High C Compiler retails for $895, and the 32-Bit Source-Level Debugger for $295. The two together are currently $995.

MetaWare Incorporated
2161 Delaware Avenue
Santa Cruz, CA 95060-5706
(408) 429-6382

INTEL 32-BIT TOOLKIT

Intel has released the Intel 386/486 C Code Builder Kit, a software development toolkit for 32-bit applications. The Kit contains the Intel 32-bit C compiler, 32-bit protected mode source-level debugger, linker, libraries, utilities, **make** facility, and DOS extender.

C Code Builder's DOS extender conforms to version 0.9 of the DOS Protected Mode Interface (DPMI), which provides access to a full 32-bit (four-gigabyte) address space through the protected mode of Intel 386 and i486 microprocessors. Support for DPMI gives developers access to DPMI-compliant, 32-bit multi-tasking operating systems such as Microsoft Windows. (The DOS extender is based on XAM from IGC.)

Developers who purchase C Code Builder are entitled to royalty-free distribution of products derived from it, as well as to customer support on Intel's toll-free hot line. C Code Builder is $695.

Intel Professional Software
Literature Center
P.O. Box 58130
3065 Bowers Avenue
Santa Clara, CA 95052-8130
(800) 548-4725

FLOWCHARTS WITH ASMFLOW

Quantasm Corporation has released ASMFLOW Professional 2.0, an updated assembly language flowcharting and source code analysis tool for MS-DOS. ASMFLOW automatically generates flowcharts and call tree diagrams from assembly language source files. Other features include stack size determination, recursion detection, CPU timing analysis, procedural cross-referencing and CPU register analysis.

ASMFLOW also offers macro expansion, conditional assembly, 80486 support, enhanced laser printing, and a larger capacity for tree diagrams. Tree diagrams can be generated from code that does not contain directives declaring the start and end of each procedure. ASMFLOW Professional 2.0 is $199.95.

Quantasm Corporation
19855 Stevens Creek Blvd.
Suite 154
Cupertino, CA 95014
(408) 244-6826

SOURCE CODE ANALYSIS

Cater Software has released Version 2.0 of C/ANALYST, a C source code analysis and documentation system. Programmers can use C/ANALYST for a high-level overview of a program. Version 2.0 supports the full ANSI C standard. Some of the new features are an ANSI-compatible token-based preprocessor, improved hashing, and full ANSI C syntax with switches for special keywords.

Enhancements to PROCTREE, the procedure tree generator, include HP Laserjet printer support, output redirection to compressed bit-mapped files, and a utility to print those files.

Version 2.0 can analyze approximately 5,000 lines/minute of C source code on a 12 MHz 286. The cost is $250 with a 30-day money-back guarantee.

Cater Software
2182 Westfarthing Way NW
Salem, OR 97304
(503) 581-5622

MULTI-PLATFORM DEVELOPMENT

The Iliad Group, Inc. is now shipping its professional software development environment, PI Edit 4.0. Version 4.0 is available for DOS, OS/2, Interactive UNIX 386/ix, SCO UNIX & XENIX, and AT & T systems. A version for IBM AIX will also be available.

New features include a TAGS function to allow C and C++ developers to reference functions and program elements directly by name, without knowing their locations. Other commands allow programmers to make, link, line, source control, debug, and execute applications with a single keystroke.

For those with UNIX backgrounds, PI adds a vi keyboard interface to "hot

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

key" between PI and vi modes. With MS-DOS versions, PI can remove itself from memory while executing other applications or the DOS shell. It can also store itself in EMS memory or on disk.

The MS-DOS version of PI lists for $195. PI for OS/2 lists for $249, and UNIX pricing begins at $349. Site licences and volume and educational discounts are available.

The Iliad Group, Inc.
77 Geary Street, Fifth Floor
San Francisco, CA 94108
(800) 473-2053

CASE:W FOR WINDOWS

CASEWORKS announces version 3.0 of CASE:W, a software engineering tool for Windows 3.0 developers. CASE:W generates all the code needed for an application's Windows interface. While no additional coding is required, programmers can append to or modify the code—generated in C—at will.

Version 3.0 adds extended code support for dialog controls such as push buttons and list boxes. CASE:W allows full access to other tools and resources necessary for applications development, such as Microsoft's Programmer's Tool Kit (PTK) and code debuggers, from within CASE:W. Applications created under CASE:W do not require special compilers or run-time systems.

CASE:W operates in a typical Windows environment: a 286- or 386-based machine with the Microsoft Windows PTK, C Compiler Version 5.1+, and the Make Utility and Linker. CASE:W 3.0 costs $795.

CASEWORKS, Inc.
One Dunwoody Park, Suite 130
Atlanta, GA 30338
(404) 399-6236 or (800) 635-1577

OOP EXTENSIONS TO C

Bogie Hardware has released OOC Libraries, a library of C functions to add an interpreted object-oriented extension language to any C program. The compact (250K) yet complete extension language, TR, supports a full compiler and byte-code interpreter, separate compilation units, C-like expression operators, data structures, hierarchical classes and methods, powerful control structures, dynamic linking of functions, methods, and data, strong typing, and inter-operability of C and TR methods, data types, classes and functions.

The OOC Libraries allow portable, object-oriented C applications to be created without special preprocessing of C source code. Full C and TR source code is provided, along with **make** files, documentation, and sample code.

The OOC Libraries are supported on Microsoft C 5.0+ under MS-DOS, SunOS 4.0 UNIX on Sun-4 platforms, and Symantec THINK C 4.0 for Apple Macintosh and DEC VAX/VMS 5.3. C and TR source that is created with OOC Libraries is portable across all supported platforms.

Single-user, single-platform licenses start at $150. Network platform pricing starts at $450. Discounts are available for multiple platform purchases.

Bogie Hardware, Inc.
502 Berlin Road
Pittsburgh, PA 15221
(412) 371-9449

WINDOWS DRIVER LIBRARY

Microsoft announces the Microsoft Windows Supplemental Driver Library, a collection of device drivers for video displays, pointing devices, and other peripherals under Windows 3.0. The SDL contains device drivers that support approximately 126 hardware peripherals with either added or enhanced support.

SDL disks can be freely copied and distributed among licensed Windows users within a company. The SDL can be downloaded from online services such as Microsoft OnLine, CompuServe, and GEnie at no charge other than connect time. In addition, the SDL is available directly from Microsoft for a nominal $20 charge to cover the cost of the four 5.25-inch disks or seven 3.5-inch disks which the SDL requires.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
(800) 426-9400

The Heap Expander™ v 2.05

Gain virtually unlimited heap space.

Function
Provides interface routines so that your programs use expanded memory, extended memory or conventional memory. As memory becomes full, The Heap Expander swaps to disk.

Utility
The Heap Expander comes with complete source code, libraries and documentation plus four utility programs that simplify development of expanded memory programs.

Guarantee
Guaranteed by The Tool Makers to perform as specified with an unlimited money-back refund.

Includes libraries and source code for all current versions of Turbo C and Microsoft C.
System requirements: IBM PC, XT, AT or clone compatible with MS-DOS or PC-DOS v. 2.0 or above.

MC/VISA/COD
800-248-1045 ext 100 USA
800-952-5560 ext 100 Idaho

still just $79.95

THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690

Circle 166 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ ARTICLE INDEX 1990

TOPIC	VOLUME	AUTHOR	TITLE
386 Development	8.3	Otken, John	386 Debugging
386 Development	8.4	Berry, Ronald	Converting to 386 Protected Mode
386 Development	8.4	Wills, Dean C.	Running Unprotected
386 Development	8.5	Baker, M. Steven	Tools for 386 DOS
386 Development	8.6	Barrenechea, Mark J.	Peak Performance: On to the 80486
ADA	8.6	Riehle, Richard	Software Components
ASM	8.1	Otken, John	Using Dynamic Memory Efficiently
ASM	8.2	Otken, John	Manipulating Large Data
ASM	8.3	Otken, John	Advanced Segments
ASM	8.4	Otken, John	Errors and Stream I/O
ASM	8.5	Otken, John	Windowing Text with X
ASM	8.6	Otken, John	Optimization Strategies
BASIC	8.2	Winer, Ethan	Fast File I/O in QuickBASIC
BASIC	8.6	Lesser, Murray L.	Calendar Conversions
C	8.1	Lesser, Murray L.	Computing Time in C
C	8.3	Naro, Rick	Benchmarking-Fact or Fiction?
C	8.4	Lesser, Murray L.	Different Times
C/C++	8.6	Holub, Allen	Virtual Memory Paging
C	8.6	Sumerlin, W.	Writing an ANSI C Preprocessor
COBOL	8.4	Riehle, Richard	Cobol-85 Nested
Commentary	8.2	Glass, Brett	The Importance of Good Defaults
Commentary	8.4	Hardenbergh, Hal W.	Japanese VCRS and DRAMS
Commentary	8.5	Hardenbergh, Hal W.	Sparcs Plot World Takeover
Commentary	8.6	Hardenbergh, Hal W.	Oh Say Can You C?
Consulting	8.1	Roetzheim, William H.	Customers You've Never Considered
Consulting	8.3	Roetzheim, William H.	Software Development '90
Consulting	8.4	Barkley, Paul	Deliberately Breaking the Rules
Consulting	8.5	Barkley, Paul	Breaking the Rules Again
Database	8.6	Glass, Brett	The Database Dilemma
Database	8.6	Curtis, Lea C.	Networking Databases
Database	8.6	Baker, M. Steven	B-Tree Indexing
DESQView	8.1	Hoff, Todd	Using the DESQview API
DOS	8.3	Greenberg, Daniel E.	Reentering the DOS Shell
DOS Extenders	8.4	Baker, M. Steven	386 DOS Extenders
FORTRAN	8.4	Baker, M. Steven	Fortran on the 386
Graphics	8.1	Abrash, Michael	Circles for the VGA
Graphics	8.2	Abrash, Michael	Faster Circles for the VGA
Graphics	8.2	Richter, Jake	Put a TIGA in Your PC
Graphics	8.2	Howard, Christopher A.	Reading the Unreadable EGA Palette
Graphics	8.2	Howard, Christopher A.	Programming the 800x600x16 Super-VGA Mode
Graphics	8.3	Abrash, Michael	What Goes Around
Graphics	8.4	Abrash, Michael	Faster Ellipses and More
Graphics	8.5	Abrash, Michael	More Colors on the VGA
Graphics	8.6	Abrash, Michael	VGA Color Paging
Hardware	8.5	Hardenbergh, Hal W.	Cache Memories
Networking	8.1	Greco, Frank D.	A Net Gain: NFS, Network File System
Networking	8.1	Schwaderer, W. David	Exploiting NetBIOS
Object-Oriented	8.5	Glass, Brett	Late Binding
Object-Oriented (C)	8.4	Holub, Allen	Using C++ Features in C
Object-Oriented (C)	8.5	Vernon, Vaughn	Writing Applications that Travel
Object-Oriented (C++)	8.5	Holub, Allen	A Virtual Memory System in C/C++
Object-Oriented (Pascal)	8.5	Swan, Tom	Pitfalls in Turbo Pacal OOP
Operating Systems	8.3	Smith, Larry	Accessing High Memory
OS/2	8.3	Little, Chip	Mouse Messages
OS/2	8.4	Glass, Brett	The World is Flat
Pascal	8.1	Swan, Tom	What's on the Menu?
Pascal	8.3	Swan, Tom	Let's List Again, and Again, ...
UNIX	8.4	Greco, Frank D.	Pink Noise
UNIX	8.6	Greco, Frank D.	Pink Noise
Windows	8.1	Hall, William S.	Dynamic Data Exchange in Windows
Windows	8.2	Hall, William S.	Communications Using Dynamic Data Exchange
Windows	8.3	Hall, William S.	Order from Chaos

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle 51 on Reader Service Card

HUGE Professional Shareware Libraries

Comprehensive, indexed collections of all known Public Domain and Shareware utilities for PC professionals, updated quarterly with the latest products. Includes extensive disk based or printed directories with multiple search options. Each library designed specifically with the PC professional in mind. Computer Language called our dBase product "*possibly the most productive $100 you'll ever spend*". VISA, MC, AmEx,POs, COD.

Product	Disks/Files	Price
dBase/dBase Compilers	56+/1100+	$99.50
AutoCAD	17+/570+	$99.50
C	19+/400+	$49.50
Turbo Pascal	23+/270+	$49.50
Ventura DTP	21+/170+	$49.50
Lotus 123	17+/200+	$49.50
Assembler	10+/170+	$49.50
Games	35+/720+	$49.50
Novell Netware	17+/170+	$49.50
WordPerfect	30+/160+	$49.50
Utilities for DOS Gurus		$49.50

EMS, 4505 Buckhurst Ct.
Olney, MD 20832 USA
(301) 924-3594 Fax: (301) 963-2708
France (1)47763872 · Germany 0911/338844
Benelux 31-20-5753019

Circle 353 on Reader Service Card

EZ-DISK COPY

Duplicates Disks Fast!

•Read diskette once, then. •Quickly, accurately mass duplicate 5.25" & 3.5" disks (360K, 720K) on your own PC/XT/AT/etc. © •Formats, copies, verifies, etc. •Even faster with pre-formatted disks. •Serialize disks and prints labels, with included utility •Replaces hardware worth $1000s.

Only $79 +S&I.

S&I: $6 USA48; $10 N.Amer; $22 Foreign

EZX, 917 Oakgrove, #101, Houston,TX 77058
Info:713.280.9900 •BBS:8180 •FAX:0525
Order (D,V,M,AX), Catalog Toll Free
1☎800•800•2468 xPJ191

Circle 352 on Reader Service Card

PC-lint

C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.

Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Contact this advertiser directly.

"QUICK & CLEAN" COBOL

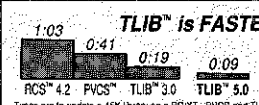
The COBOL Report Activator (COBRA):

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and COPY members
- Produces audit trail reports
- Up to 5 output files per program
- Provides report options & site defined defaults
- Is available for Mainframe & PC

COBRA
Systems Software
David Olivadoti, P.O. Box 750817
Petaluma, CA 94975
(707) 763-5123

Circle 281 on Reader Service Card

TLIB™ is FASTEST!



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, Computer Language

"TLIB is a great system" J. Vallino, PC Tech J

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with **Opus™ MAKE** & **Stick™ MAKE**.

MS-DOS $139, OS/2 $195 + shipping Visa/MC. 5 station LAN license $419 (OS/2 $595); call for other sizes.

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 (919) 233-8128

Circle 288 on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle 320 on Reader Service Card

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672
802-253-4437

Contact this advertiser directly.

FORTRAN

FORTRAN Toolkits
Comprehensive libraries of over 300 graphics, text, and systems service subroutines from ellipses to child-processes to I/O verification. High-speed video drivers, from CGA to Super VGA, are all internal. Option rich, easy to use. No royalties. 300+ pg. manual, 8,000 lines of sample source. **Price $215.**

DOCUMENT'er and DIAGRAM'er
Makes source code understandable. Indexes, cross-references, and diagrams the structure of source code. No changes to source required. Nested includes, recursion, and annotations handled automatically. **Price $165.**

IMPULSE *Engineering*
PO Box 190206,
San Francisco, CA 94119-0206
(415) 788-4611

Circle 84 on Reader Service Card

THE CD-ROM DEVELOPER'S LAB™

Comprehensive CD-ROM multimedia production resource for PC and Apple developers. Proven design, management, data prep, programming, premastering and manufacturing techniques. Important specs from leading companies. Demos of off-the-shelf tools for imaging, audio and animation. Sample applications demonstrate the power of our source tools in C, Turbo Pascal.
PC or MAC @ $395.00;
Transportable version @ $425.00,
Visa/MC accepted.

SOFTWARE MART, INC.
3933 Spicewood Springs Rd., Suite E-100
Austin, Texas 78759
(512) 346-7887 FAX: (512) 346-1393

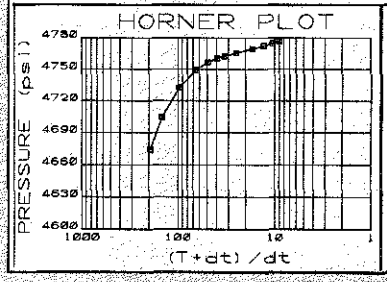
Circle 79 on Reader Service Card

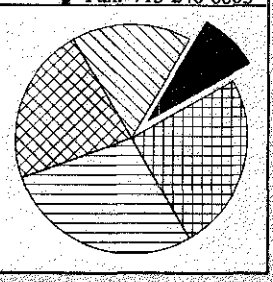
INGRAF 5.00

A Multi-Device Graphics Library for Scientific, Engineering and Business applications. Supports Video(CGA, EGA, VGA & Hercules), Printers (EPSON, Okidata, HP LaserJet, HP Paintjet & Postscript) and Pen Plotters(HPGL and DM/PL).

C, FORTRAN-77, PASCAL, MS Professional BASIC and QuickBASIC bindings.

FULL SOURCE CODE · NO ROYALTIES · $350.00





Sutrasoft
10506 Permian Dr.
P.O. Box 1733
Sugar Land
TX 77487-1733
U.S.A.
Tel: 713-491-2088
Fax: 713-240-6883

Circle 253 on Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Regular listings include a headline, company information, and approximately 400 characters for your message. Display ads are 1 5/8" wide by 2 3/8" high. For more information or to reserve a space, call our ad department at **503-747-0800.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle 16 on Reader Service Card

sPORTt
serial PORT tester

sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.

- displays all data going in and out in ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $89.95!

WHS
3037 Grass Valley Hwy #8201
Auburn, CA 95603
916-885-2480

Circle 329 on Reader Service Card

BGI PRINTER DRIVERS
Full Source Included!

The BGI Printer Driver Toolkit provides BGI printer drivers for Epson 9 & 24 pin dot matrix, HP LaserJet II, and HP PaintJet printers. Use the BGI graphics interface to effortlessly generate high resolution hardcopy with Turbo C, C++, and Turbo Pascal. Full driver source provided for your tutorial pleasure. $89.95 ppd USA, elsewhere add $4.00. VISA & MC accepted..

RYLE DESIGN
PO Box 22, Mt. Pleasant MI 48804
517-773-0587

Circle 210 on Reader Service Card

The $25 Network
Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

iMODES Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Circle 348 on Reader Service Card

QUICKBASIC PDS 7/NETWORK
B+ TREE FILE INDEXING

Maintain B+ Tree file indexes in your QuickBASIC/PDS 7/NETWORK programs with INDEX MANAGER. Access files randomly by full or partial key, or sorted forward or backward. One module performs all functions, making IM very easy to use. Only indexes managed! You retain full control over your data. Assembler code and cache buffers make IM very fast. Includes sample programs and User's Guide. Only $59 (+tax in CA). PDS 7 version $99. Network version $250.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
619-440-6482

Circle 335 on Reader Service Card

Software Marketing

The *Software Success Reference Book (1987-1988)* is a **MUST READ** if you want to market your software products successfully. Written by David H. Bowen, publisher of *Software Success,*™ the monthly newsletter on successfully running a software business. The *Reference Book* is a 268-page guide organized by topic. Covers Lead Generation, Promotion, Pricing, Distribution, Support, etc. **Only $25.** Check or Credit Card (VISA/MC/AEX).

100% Money Back Guarantee

Software Success
PO Box 9006, San Jose, CA 95157
(408) 446-2504
FAX (408) 255-1098

Circle 67 on Reader Service Card

Developer's Office Management System

Easy to use computerized system to organize, operate and control your software business.

Major features include:
- User and Dealer indexed database
- "Bug" look-up DB for software support
- Invoicing capabilities
- Utilities i.e., serial # embedding program
- Directory of packagers, suppliers, etc.
- Directory for product releases
- Lots of reports and other features
- Free 900 number to support your software

$69.00 (+ tax in Texas)
(VISA/MasterCard)

512-834-8927

S.L.A.M. Software
3202 W. Anderson Ln. Ste. 208-176
Austin, TX 78757-1022

Circle 12 on Reader Service Card

UPPER DECK FORTH
$49

Based on the Forth-83 Standard. Fully segmented architecture. Fast compile, no link required. Uses normal DOS files. Includes editor, assembler, decompiler and debugger. Turnkey application support. For PC's and compatibles, 256K, DOS 2.0 or later. Add $3 for shipping and handling (outside USA $15). CA residents add sales tax.

UPPER DECK SYSTEMS
P.O. Box 263342
Escondido, CA 92026
(619) 741-1075

Circle 24 on Reader Service Card

BLUE JAY SYSTEMS

the EDitor, a quality full feature programmer's editor. A professional tool, it simplifies development and debugging of all software. Quickly and easily learned. Full screen text, fast, multiple files and buffers, Undo delete, text pattern search and replace, text moves between multiple buffers and files, optional line numbers, extended ASCII characters, in-line DOS commands and more. Comes with concise wire bound manual. VGA, EGA, CGA. IBM & compatibles. Why pay more than $87? Available from: BLUE JAY SYSTEMS, PO Box 608, Mount Vision, NY 13810 (607) 293-8007

Circle 11 on Reader Service Card

A Software Development Essential!

DOCZ™

Save up to 35% of your development time with DOCZ to automate subroutine and program documentation. Take advantage of the object-oriented technology of DOCZ to share and re-use your software modules. FREE MSDOS demo disk and samples. 30-day money-back guarantee. VISA/MC accepted. $195/Single-User. Start benefiting NOW!

Do It Better With Toolz!

Software Toolz, Inc.
8030 Pooles Mill Drive
Ball Ground, GA 30107-9610
(404) 889-8264
Orders, V-mail: (800) 869-3878

Circle 78 on Reader Service Card

INTEL® 80386 MACRO DISASSEMBLER

MD86 (Masterful Disassembler) is the most comprehensive macro disassembler on the market. Interactive by design not after-thought. Features include:

- ► User defined macros with arguments
- ► Auto code/data separation
- ► Auto commenting and label naming
- ► Command menus and help screens
- ► Customizable disassembly options
- ► Comprehensive, indexed, manuals
- ► Fully supports 8086/87 thru 80386/87
- ► COM/EXE/SYS/memory/other
- ► Cross reference and more!

Still only $67.50 + tax ($1.50 s/h)

C.C. SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Circle 132 on Reader Service Card

Coming Next Issue:

LANGUAGES

March/April 1991 Vol. 9.2
Advertising space reservations are due:
January 4, 1991
Materials are due:
January 11, 1991

Please call 800-234-0386.
Ask for Jay or Robin in the advertising department to reserve your space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
345	Alphax Systems, Inc.	18
118	Artisoft Inc.	C3
30	Austin Code Works, The	53
51	Az-Tech Software, Inc.	93
343	Base Two Development	81
14	Blaise Computing, Inc.	54
11	Blue Jay Systems	94
3	Borland International	C4
288	Burton Systems Software	93
336	C Source	68
132	C.C. Software	94
321	Catenary Systems	31
335	CDP Consultants	94
53	Cnapse	95
281	Cobra Systems Software	93
39	Coleman Softlabs	5
229	Disk Software, Inc.	29
353	EMS	93
352	EZX	93
261	Genus Microprogramming, Inc.	27
*	Gimpel Software	93
18	Goin Places Software	12
21	Greenleaf Software, Inc.	57
56	Greenview	41
84	Impulse Engineering	93
348	Information Modes	94
46	InSync Software	95
50	INTEL	29
317	Island Systems	39
211	Kadak Products, Ltd.	45
295	Knowledge Dynamics Corp.	8
63	Laboratory Microsystems, Inc.	88
44	Lahey Computer Systems, Inc.	75
246	Magee Enterprises, Inc.	58
*	Mark Williams Company	13
220	Micro Security Systems	34
232	Microsoft—Human Resources	15
40	Microsystems Software	22
266	Mix Software	59
94	MMC AD Systems	36
77	Modular Software Systems	17
109	Nanosoft	74
160	Nu-Mega Technologies	2
16	Opt-Tech	94
254	Opus Software	64
270	OTG Systems, Inc.	64
208	PC Tech, Inc.	31
36	PC Tech, Inc.	74
6	Periscope Company, Inc., The	71
*	***PJ* 1990 Article Index**	92
*	***PJ* Bulletin Boards**	90
*	***PJ* Mail Order Store**	80
279	Pocket Soft, Inc.	9
54	Programmer's Corner, The	8
48	Programmers Warehouse	11
60	QNE International	55
*	Quaid Software Limited	37
252	Quantasm Corporation	66
147	Quarterdeck Office Systems	C2
147	Quarterdeck Office Systems	1
20	Rational Systems, Inc.	49
59	RJSwantek & Associates	7
210	Ryle Design	94
12	S.L.A.M. Software	94
65	Schroff Development	31
269	Sequiter Software Inc.	63
165	SLR Systems	65
167	Software Development '91	25
79	Software Mart	93
67	Software Success	94
78	Software Toolz, Inc.	94
110	Softway, Inc.	28
82	Softworks	26
47	Solution Systems	19
10	South Mountain Software	69
330	Stanwood Associates, Inc.	40
68	Sub Systems	34
253	Sutrasoft	93
81	Symmetry Group, The	30
166	Tool Makers, The	91
327	Tools & Techniques, Inc.	18
276	Tri-Technology Systems, Inc.	60
72	TSR Systems	95
25	TurboPower Software	23
24	Upper Deck Systems	94
125	V Communications, Inc.	79
*	Vermont Database Corp.	93
320	Wenham Software Co.	93
100	Western Wares	60
329	WHS Western Hydro Systems	94
55	Winsoft	26
75	Zircel Software	56

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

Real-time programming is tough... unless you have SynRTX

SynRTX is the most comprehensive **object-based Real-Time eXecutive library** (where each object is a task) available for C and C++. It allows your PC to perform hundreds of tasks simultaneously at a simplicity you just wouldn't believe. A complete reference manual and application examples show you how to make it fly.

No charge for AIR shipping anywhere in the world. 30-day money-back guarantee. 1-year upgrade. No royalties, and free technical support.

Low price, simplicity, speed, and service. It's a combination you can't afford to miss. So act today -- at $79, there's no reason to wait!

If you **call our fax NOW,** our free brochure that explains all the confusion with today's real-time buzz words will be mailed to you within 24 hours.

Get also our **real-time malloc package:** the fastest memory allocation/deallocation based on object's lifetime. Replace your malloc, and speed up your program, for just $29.

Cnapse The Real-Time Contact
(514)447-7221 Fax:(514) 447-7297
11 Laurier, Suite 103, Chambly, Qc, Canada J3L 9Z7

Circle 53 on Reader Service Card

QUALITY TOOLS

Memory Overlay Manager - transform your C, Clipper or Clarion application into a 5K TSR with hot-keys.....$99.95
Memory Master - break 640K barrier, one heap or many, EMS/XMS.....$59.95
ASMplus - structured ASM lang. & lib. Code ASM just like C or Pascal.....$99.95
"C"erious Toolkit Plus - over 250 highly optimized fcts, C & ASM source...$99.95
386PM protected mode ver.....$199.95
Phase One - 100% replaceable plug-&-play C engine for Paradox, runs 4-10X faster, DOS and DDL ver. incl.....$495.00
PALCom - PAL lang. compiler. Create stand alone .EXE programs.....$1495.00

Please call our BBS at 516-331-6682 to try out our FREE demo programs.

TSR Systems Limited
116 Oakland Avenue
Port Jefferson, NY 11777
Ph: 516-331-6336 FAX: 516-331-6377

Circle 72 on Reader Service Card

ATTENTION Software Writers!

Is your software innovative, usable and needed? Are your applications practical, thorough and well tested? Is your documentation well organized and understandable? InSync, a division of an established international software distributor, is evaluating new and innovative software to publish and market domestically and internationally. We are seeking new off-the-shelf MS/PC-DOS software applications (no accounting or inventory, please). ORIGINALITY IS A MUST for all submissions.

Please call our Evaluations Department to discuss submission procedures.

InSync software
1-800-783-7053

Circle 46 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Goliath Beats David—As Usual

By Hal W. Hardenbergh

As I write this, Intel's 386DX, 386SX, 486, and i860 are all on allocation, despite the fact that Intel will produce nine million 386 chips this year. Sun Microsystems and MIPS, generally considered the leading RISC houses, both have a hiring freeze and are cutting back expenses wherever possible. MIPS stock dropped from 27 3/4 to 7 3/4 in a 12-week period. Intergraph's RISC chip, the old Fairchild Clipper, has been officially withdrawn from the UNIX merchant market. Motorola's 88000 has one foot in the grave and the other on a banana peel.

I'm going to reveal a dirty little secret, one which lots of industry "experts" are trying to cover up: RISC has utterly and completely failed. It has died. The corpse turned cold about late summer of this year, and it's beginning to stink as winter approaches.

RISC was built on three basic philosophies: a small, simple instruction set; a small chip size; and a very fast external memory system. All of these philosophies are now bankrupt.

It seemed obvious to me from the beginning that the use of 32 multiply-step instructions to replace a multiply instruction was a big mistake. I've seen SPARC-based boxes used for image processing, which uses *lots* of integer multiplies. The programmers who have worked with both Motorola CISC-based machines and SPARC boxes tell me that the SPARC boxes are *incredibly* slow when doing multiplies. The Sun compiler generates horribly inefficient code, far worse than a simple sequence of 32 multiply-steps. Guess what? Sun's latest RISC assembler release includes an integer multiply instruction, obviously for an as-yet-unreleased version of the SPARC chip.

To MIPS's credit, the MIPS RISC version broke with the RISC philosophy from the very beginning by using special (complex and microprogrammed) circuitry to perform multiplies. But what really broke the back of the simple instruction set philosophy was IBM's release of a desktop computer early this year which was *provably* superior to the best SPARC and MIPS computers, and which uses a complex repertoire of 184 instructions, including two flavors of string search instructions. More recently, the complex i860 has also proved to be far better than the SPARC or MIPS running the SPEC benchmarks. **Exit small, simple instruction sets.**

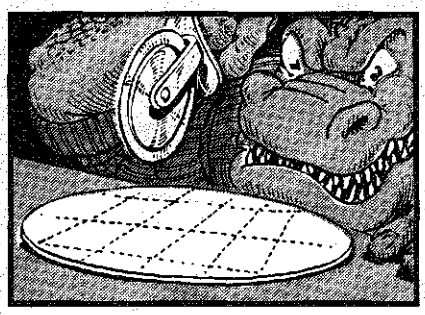


ILLUSTRATION BY JIM CARPENTER

John Hennessy and David Patterson, closely associated with the MIPS and SPARC architectures, have collaborated on an excellent book on computer architecture. This 1990-copyright book extols the virtues of small CPU chips. Page 56 shows a six-inch wafer containing 80 i486 dies, and page 57 shows a six-inch wafer containing 246 Cypress SPARC dies. Additional photos and text hammer home the superiority of the small RISC die.

Guess what? At the end of 1990 *every* hardware designer working with microprocessors knows that the next generation needs multiple execution units to achieve more than one operation per clock. This automatically requires some complex scheduling and scoreboarding circuitry in order to keep track of data dependencies. *Every* upcoming high-performance microprocessor will use multiple execution units. **Exit small, cheap CPU dies.**

The current generation of microprocessors runs at 50 MHz. Motorola's 68030 has been available at that speed for some time, and the 50 MHz 486 will be shipping in 1991. (The rumors I've heard say that Intel has been making 50 MHz 486s for a while, but does not yet have production test gear that will run that fast.) Clock speeds of 100 MHz are being seriously considered for the next generation of CPUs.

This means that it is no longer possible to build an external memory system which can keep up with those multiple execution units running at 50 to 100 MHz. It's not just difficult, it's impossible. The only solution currently available is to use on-chip cache memory, so that most memory transactions occur on-chip and not to the slow external memory system. That's why Intel now makes three microprocessors (including the new 960MM) which use more than one million transistors, and why the second-generation i860 will use 2.5 million transistors. **Exit small and medium CPU dies.**

The RISC camp has an additional problem in that UNIX uses frequent context switches and so requires much larger caches for a given level of performance. The 486 and i860 are most often used in single-tasking environments and therefore can provide outstanding performance using a "mere" 1.2 million transistors!

Every next-generation microprocessor will have multiple execution units, specialized circuitry for math, and a large on-board cache. They will *all* be very large, highly complex, and not at all cheap. RISC is dead, dead, dead.

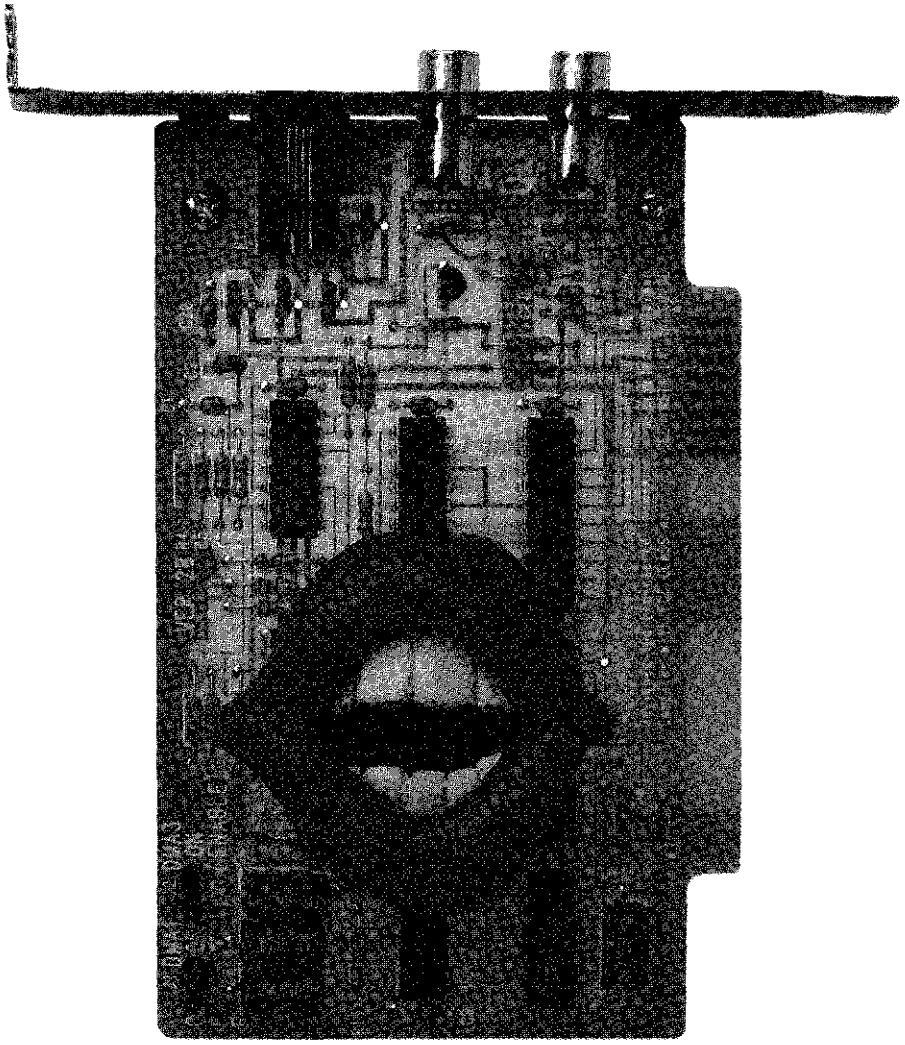
What are the RISCites going to do? They're going to assert that these very large, complex, expensive chips are RISCs, and try to keep a straight face.

To design and build such chips requires a great deal of money. In its fall quarter, Intel sales were over $1 *billion*, its profits were $172 million, its cash reserves were $2.3 billion, and it now produces one million 32-bit CPUs (including the 386SX) a month. Intel spent $135 million on R&D that quarter, which means that Intel is now investing $1.2 *billion* in R&D per microprocessor generation.

How would you like to own stock in one of the 22 companies which are trying to compete with that juggernaut by selling an average of 1,000 "RISC" chips per month? PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Make your software laugh, sob, growl, warble, wail, sing and talk.



If you've ever wanted to add sound to your software, put on your headphones. The LANtastic™Voice Programmers Interface (VPI) and the LANtastic Voice Adapter are waiting.

Now you can create educational and training programs on disk or CD-ROM that actually speak to students.

Or voice mail packages that send speech across a LAN.

Or "talking" software demos.

Music synthesis.

Sound effects.

And whatever else echoes through your imagination.

You see, the LANtastic Voice Adapter is a low-cost expansion device that converts between audio signals and a digital data stream. It has mono tape in and out jacks so you can record from a CD player, tape recorder, microphone or other external sound source, then play back through an amplifier with speakers, or through the telephone handset provided with each adapter.

Better yet, the LANtastic VPI is a set of simple, straight-forward software tools, which make it possible for your applications programs to use the LANtastic Voice card to record and play digitized sound. And no one else offers anything like it. Especially at $149* per adapter and $195 for the VPI.

No wonder *PC Magazine* (July 1990) wrote: "Leonardo DaVinci might have called it a work of art." *And :* "LANtastic's strong VPI makes it a natural choice for developers wishing to add another dimension to their communications."

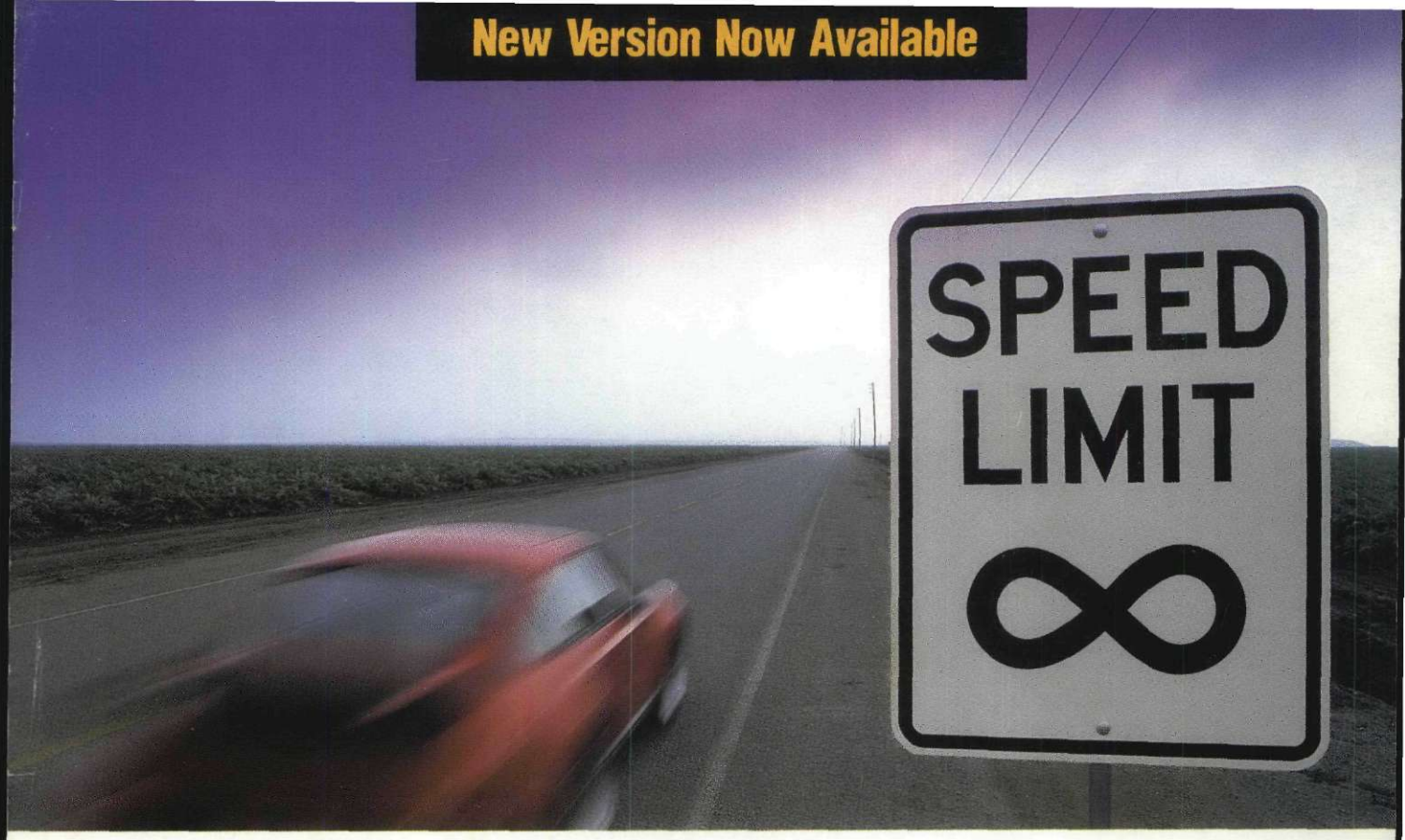
For complete information about Artisoft's Voice Developers Kit, call 602-293-6363.

ARTISOFT

*Manufacturer's suggested retail price. ©1990 ARTISOFT. LANtastic is a trademark of Artisoft. Brand names and product names are trademarks or registered trademarks of their respective companies.

Circle 118 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Borland's Turbo Pascal 6.0 is the Fastest Way from Inspiration to Application

Jump-Start Your Application

When you're inspired to write a program, you want to spend your time developing code that solves your problems. Not hours and hours writing common routines for event handling, data management or user interface.

Now, you can jump-start your applications development by programming with the latest release of the World's #1 Pascal Compiler, Turbo Pascal® 6.0 with Turbo Vision.™

Now with Turbo Vision

With Turbo Vision, the first object-oriented application framework for DOS, you get a giant head start on creating better applications in far less time. Use a Turbo Vision object and your program automatically inherits a hot program architecture that includes overlapping windows, pull-down menus and mouse support. Turbo Vision makes it fast and easy—setting you free to develop the parts of your applications that solve your problem.

And Turbo Pascal 6.0 comes loaded with Turbo Vision applications including a calendar, a calculator, an editor, a clock, a directory browser and forms.

New Turbo-Charged Environment

The new Turbo Integrated Development Environment (IDE) features a multi-file editor, overlapping windows and mouse support.

And Turbo Help lets you copy, compile and run an example program for every standard Pascal library routine so you can use it in your code.

Pro Version with Turbo Drive™

Turbo Pascal® Professional 6.0 also includes: a professional version of the compiler with Turbo Drive,† for compiling big applications in extended memory; Turbo Debugger® 2.0, for killing the toughest bugs; Turbo Profiler™ 1.0, for eliminating bottlenecks; and Turbo Assembler® 2.0, the world's fastest, 100% MASM-compatible assembler.

An Inspiring Offer

The suggested retail price for Turbo Pascal 6.0 is 149^{95}, and 299^{95} for Turbo Pascal Professional 6.0.

REGISTERED TURBO PASCAL OWNERS. You can upgrade to Turbo Pascal 6.0 for only 69^{95}* or to Turbo Pascal® Professional 6.0 for only 99^{95}* direct from Borland.

To order, see your dealer
To upgrade, call now: 1-800-331-0877

BORLAND

The Leader in Object-Oriented Programming

Circle 3 on Reader Service Card

CODE: MA92

*Add $10.00 for shipping and handling. Offer expires February 28, 1991. This upgrade offer good in U.S. and Canada only. Mail orders to: Borland International, Inc., P.O. Box 660001, Scotts Valley, CA 95067-0001. Residents in CA, CT, GA, IL, MA, MI, NY, OH, PA, TX, VA and WA, please add appropriate sales tax. For orders outside the U.S., call (408) 438-5300. †Turbo Drive compiler requires 1Mb extended memory. Hard disk recommended. Copyright ©1990 Borland. All Borland products are trademarks of Borland International, Inc. Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Denmark, France, Germany, Italy, Japan, Sweden and the United Kingdom. Porsche is a registered trademark of Porsche. BI 1392

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED